Spring 2019

A Partially Automated Process For the Generation of Believable Human Behaviors

Bridgette Parsons

A Partially Automated Process For the Generation of Believable Human Behaviors

by

Bridgette Parsons

Bachelor of Science
California University of Pennsylvania 2003

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2019

Accepted by:

Jose M. Vidal, Major Professor

Michael Huhns, Committee Member

John Rose, Committee Member

Marco Valtorta, Committee Member

Abbas Tavakoli, Committee Member

Cheryl L. Addy, Vice Provost and Dean of the Graduate School

Dedication

For my family, both human and furry, who have helped me achieve what I have wanted to do since I was in Middle School. Without you, I would have given up long ago.

Acknowledgments

I would like to thank my committee, Dr. Huhns, Dr. Rose, Dr. Valtorta, and Dr. Tavakoli, for supporting and assisting me throughout my many years of research. I would particularly like to thank my advisor, Dr. José Vidal, for his guidance and encouragement, and for his patience. This has been a much longer journey than I anticipated, and it would not have been possible to complete without their help.

I would also like to thank my friends and study partners, Dr. J.J. Shepherd, Dr. Richard Walker, Jay Swain, and Chris Schmidt, for making it easier to return to school so many years after graduating. They made studying more enjoyable, and were all helpful with their abilities to see problems in different ways. Together, we were able to solve things that were much more confusing to attempt on our own.

Finally, I would like to thank my family for always being so helpful and supportive. My parents have always emphasized the importance of education, and gave up a great deal to make sure that my sister and I would be able to reach our educational goals. My sister has always been insistant on my ability to succeed, even when I didn't believe in myself. My cousin, Dr. Christine Stephenson, frequently commisserated with me about graduate school. She was able to make me see the funny side of the things that I found frustrating, making it easier to continue my work. My husband, Kevin, has been especially supportive and understanding. Without him, I never would have been able to even return to school, much less finish.

Abstract

Modeling believable human behavior for use in simulations is a difficult task. It requires a great deal of time, and frequently requires coordination between members of different disciplines. In our research, we propose a method of partially automating the process, reducing the time it takes to create the model, and more easily allowing domain experts that are not programmers to adjust the models as necessary. Using Agent-Based modeling, we present MAGIC (Models Automatically Generated from Information Collected), an algorithm designed to automatically find points in the model's decision process that require interaction with other agents or with the simulation envionment and create a decision graph that contains the agent's behavior pattern based upon raw data composed of time-sequential observations. We also present an alternative to the traditional Markov Decision Process that allows actions to be completed until a set condition is met, and a tool to allow domain experts to easily adjust the resulting models as needed. After testing the accuracy of our algorithm using synthetic data, we show the results of this process when it is used in a real-world simulation based upon a study of the medical administration process in hospitals conducted by the University of South Carolina's Healthcare Process Redesign Center.

In the healthcare study, it was necessary for the nurses to follow a very consistent process. In order to show the ability to use our algorithm in a variety of situations, we create a video game and record players' movements. However, unlike the nursing simulation, the environment in the game simulation is more prone to changes that limit the appropriate set of actions taken by the humans being modeled. In order

to account for the changes in the simulation, we present a simple method using the addition of a hierarchy of rules with our previous algorithm to limit the actions taken by the agent to ones that are appropriate for the current situation.

In both the healthcare study and the video game, we find that there are multiple distinct patterns of behavior. As a single model would not accurately represent the behavior of all of the humans in the studies, we present a simple method of classifying the behavior of individuals using the decision graphs created by our algorithm. We then use our algorithm to create models for each cluster of behaviors, producing multiple models from one set of observational data.

Believability is highly subjective. In our research, we present methods to partially automate the process of producing believable human agents, and test our results with real-world data using focus groups and a pseudo-Turing test. Our findings show that under the right conditions, it is possible to partially automate the modeling of human decision processes, but ultimately, believability is greatly dependent upon the similarity between the viewer and the humans being modeled.

Table of Contents

List of Tables

List of Figures

Introduction

From the time that Alan Turing published his paper on the imitation game [86], people have been attemping to make computers behave in a manner that is indistinguishable from humans. While some of the issues mentioned by the early detractors have been resolved, new problems have more recently come to light. Believable imitation of humans by a mechanical construct is a difficult endeavor, and it is one that people have tried to solve in many different manners.

What is believable? Although there have been many attempts to make machines appear human, we are not only hampered by accuracy. In some instances, as is the case in graphics or the visualization of physics-based reactions, such as collision, what many humans think is realistic behavior is not necessarily an accurate representation of the real world. Believable human behavior is behavior that appears to be human to human observers, even if it is not completely accurate real-world behavior. A believable human agent is one that appears to act like a human in a given situation. Whether or not the agent is believable may vary based not only on the agent, but also on the observer. What is believable to one human may not be believable to another.

Early chatbots, programs that respond to user input with textual conversation, were limited to scripts that responded based upon keywords. Programs like ELIZA [92], a machine imitation of a psychologist, and PARRY [17, 18], an imitation of a paranoid schizophrenic, were interesting as scripted examples, but were unlikely to be mistaken for humans. Robots, with their necessity for redundant programming in case of mechanical malfunction [90], were even more limited. Hardware limitations

necessitated the use of fewer resources for decision-making, requiring AI programmers to limit their algorithms in order to function with what was available.

Despite improvements in computer architecture, the difficulty in the believable imitation of human behavior remains. For obvious reasons, direct imitation of the inner workings of the human brain would require excessive time and space complexity. As the human brain is composed of an estimated 10^{11} neurons, which may have 10^{15} connections between them at any given time, and multiple connections being used simultaneously for a single brain function [83], physical imitation of the human brain is a task too great for us to accomplish. Various neural imaging techniques have shown that, not only are multiple regions of the brain used for various tasks, but the exact regions vary between individuals, further complicating the process. The variance between individuals and the sheer computational power needed to directly imitate the working of any given human brain make physical imitation an unlikely method to produce believable human behavior.

Thus limited by hardware and biology, programmers have turned to other methods of imitation. Following the work of cognitive psychologists and cognitive scientists, there are those who attempt to imitate the overall processes of human thought. Simon and Newell created a system of physical symbols to describe intelligent action, but were limited by the amount of processing power and storage necessary, and the heuristics that were vital to make their methodology work beyond a basic level [64]. Minsky's *Society of Mind* [59] was another good illustration of the complexity of this task. Although the theories presented provide a useful context, they remain highly abstracted, and attempts to implement the structures represented in Minsky's work have been severely limited due to their complexity [79]. While multi-agent systems is mentioned as a possible method for the implementation of Minsky's framework, it has yet to be achieved.

While the imitation of human thought remains a daunting task, it does not appear to be necessary for the imitation of human behavior in a simulation. It is not necessary for a machine to think like a human would think in order to do what a human would do. In an attempt to simplify the problem, we have turned to imitation learning - a methodology based upon the recognition of the patterns of behavior of similar humans in a given environment. In this way, we attempt to provide a partially automated process to create a model of human behavior that acts correctly in a simulation of that environment, making it easier for developers and domain experts to evaluate and adjust the simulation as necessary, and avoiding the inherent complexity of modeling human cognition.

Although it is possible to calculate by hand the probabilities necessary for a simulated human to make decisions, it is difficult and painstaking work. The difficulty of such a calculation is increased proportionately to the number of tasks that must be simulated, and is made even more difficult if the modeler is not also a domain expert. This increases the potential for error, and also the time necessary to complete the model. By partially automating this process, we are able to reduce potential error, and increase the speed at which the model is created. Partial automation makes it simpler for both the modeler and the domain expert, allowing them time to adjust the model as they wish.

In order to show that our process works in multiple domains, we apply it to both a medical administration setting and a first-person shooter video game, two domains that differ greatly. We also show that it is possible to further reduce error by automatically recording observations, rather than relying upon human observers. Human agents appear in many different types of simulations, some of which contain large-scale environmental changes. To adjust to changes in the simulation environment, we add an additional level to the decision-making hierarchy. This simple addition enables the agent to respond to changes in the environment by limiting its actions to

ones that are appropriate in the current environment, making the agent appear more believably human.

Using a distance metric, we are able to group humans with similar behavior patterns. In simulations with large data sets that comprise the actions of several humans, it is possible that there is more than one behavior pattern to be modeled. If we use all of the observations at once, the resulting average behavior may not reflect the behavior of anyone in the data set. The groups we create using our distance metric can be used to create multiple models from the same data set, providing agents with differing behaviors that more properly reflect the behaviors of the humans that were observed.

By modeling what we can directly observe, we simplify the process of creating believable human agents. Partially automating this process increases the speed at which in can be accomplished, reduces error, and allows modelers and domain experts the opportunity to more easily change the model. This, in turn, provides the opportunity to observe the impact of behavioral adjustments without the necessity of making changes in the real world.

Chapter 1

Background and Previous Work

Agent-based modeling is a useful method to simulate systems in which individual behaviors are varied and complex. It has the ability to show phenomena that arise from the interaction of individual agents that would otherwise be difficult to predict. It is also a flexible system of simulation, allowing the programmer to more easily change the behaviors of individual agents, or to add more agents to the system. This makes it ideal for the simulation of complex, dynamic systems, particularly those that involve human behavior [12].

Because simulation has become such a commonly used tool, it is important that it is both reliable and valid. A simulation can be said to be reliable if it produces the same results when given the same data. While it can be relatively easy to determine if there are consistent results that are obtained with a particular model, determining the validity of that model can be more difficult [27]. The validity of a simulation can be defined in more than one way. If a simulation appears to observers to respond in the same way as what is being simulated, it can be said to have "face validity." This is the easiest type of validity to determine, but it can also be inaccurate. "Event validity", obtained by providing the simulation with known data and checking the results, is a more accurate method of determining the validity of the simulation. By using known data, the correlation between the results of the simulation and the known results can be determined, giving a good sense of the simulation's accuracy [84].

1.1 Modeling Human Behavior With Agent-Based Models

There have been many different decision-making methods used in the attempt to create believable behavior in agent-based models. Some researchers, such as Konolige, have argued in favor of a deductive, first-order logic system of decision-making, basing the agents' knowledge on a core set of beliefs and the knowledge that can be derived from it [51]. In this type of model, the agent's behavior is rule-based, using only the knowledge that the agent has about the world and what can be logically determined by using that knowledge. This, however, does not appear to be the way that humans reason, and seems more suitable to a knowledge base than an agent-based model.

To make the model less restrictive, some researchers have used Hintikka's "possible worlds" approach, allowing the agent a certain amount of belief in any state of the current world that could be possible, based on the facts the agent is currently aware of [41]. Ginsberg and Smith argued for this approach, because the number of changes in the world between time steps is relatively small, so that keeping track of changes in the world, and therefore changes in the agent's belief state, should be relatively simple [30]. However, when using their approach, if information contradictory to an agent's current belief state is discovered, the entire knowledge base needs to be reconstructed. It then becomes extremely computationally expensive. Furthermore, scalability becomes an issue, as beliefs about every possible fact in the given world have to be stored and updated for each agent individually.

The BDI model, or Belief, Desire, and Intentions model, is a less computationally complex version of the possible worlds model. It stores the agent's current beliefs in the state of the world based upon past events of which the agent is aware. Desires are the agent's current goals, and Intentions are the agent's plans to achieve those goals. This model is based upon ideas from cognitive psychology about the nature of human thought [28]. By extending intentions to include multi-agent planning, Cohen and Levesque described a method of using BDI models in multi-agent systems [16].

Their agents also include a degree of commitment to their goals, allowing an agent to drop its goals in favor of a new one when interacting with another agent.

1.2 Believable Agents in Games and Simulations

There is a difference between an agent that acts in a believable manner and an agent that acts in an optimal manner. Human behavior has been shown to be not entirely rational, and can be influenced by many factors, including emotion, personality, cultural bias, social bias, and other factors [65]. Therefore, believable human agents cannot be modeled in the same manner as optimal agents. In order to be "believable", the agent must act in a way that is not always optimal in order to appear convincingly human.

Believability is also affected by the target audience [37]. If the agent acts in a manner that is very different from the actions the audience would take, it will appear less believable to them. This means that the target audience must be taken into account when building a human agent, since the agent will appear more believable if it is based upon the actions of people similar to the target audience. The concept of similarity to the target audience making a simulation more believable has been used for a long time by marketers [31] [10], and it has more recently been shown to be more effective in games and simulations as well [1]. The study of gamer reactions to agents based upon player input versus that of agents designed to make optimal decisions has shown that suboptimal agents that react like players are considered more believable.

The believability of a human agent is generally determined using "face validity", as mentioned in Section 1.1. While the Turing test [86], or "Imitation Game", was considered for a long time to be the ultimate test of whether or not an agent appeared to be human, it has more recently been determined to be a hindrance rather than an accurate goal [25]. Focus groups and variations of the original Turing test have become more common. There are still tests that contain human players and agents,

where testers are asked to guess which is the agent and which is the human, as is the case in the Botprize competition described in Section 1.5. However, there are also tests where videos of agents are shown to testers, and they are asked which agent appears more human, and why that is the case. There are also tests which ask testers to rate on a numeric scale how human the "agent", which may or may not actually be a human, appears to be [57].

1.3 Stochastic Models, Chaotic Systems, and Multi-Agent Systems

Stochastic models are used to indicate a degree of uncertainty in the model. Models that include human behavior must account for variance in that behavior, and therefore include some degree of uncertainty. There are various ways of modeling uncertainty by using conditional expectations which are mapped to random variables, including stochastic Markov chains and processes [13], Bayesian Generalized Likelihood Uncertainty Estimation [11], approximate entropy [73], and Agent-Based Modeling [71].

Chaotic systems are considered a subset of stochastic systems. In chaotic systems, interaction between the components of the system makes a small change in input to one portion of the system produce a large change in output. This can be seen in models of large-scale, interdependent systems, such as weather models, economic models, and sociologic models. [2]

Emergent properties, which are properties that arise from the interaction between system components but are not a part of those components, are a by-product of chaotic systems. While emergence is separate from self-organization, the two are frequently linked [38]. This makes multi-agent systems an effective way to model chaotic systems [21]. They provide a simple method of modeling complex systems by breaking them into smaller subcomponents with more simple rules. The results, and

emergent properties of the system, are a by-product of the interaction between the components.

Human behavior is inherently complex, which is why it has been modeled as a chaotic system [5]. This makes a model containing interaction between humans a complex system of complex systems. Agent-based modeling is useful to model human behavior, as it simplifies the modeling process by reducing the difficulty of modeling the individual components of the system, therefore reducing the difficulty of modeling the system as a whole.

1.4 Sequential Observations Versus Cognitive Modeling

A problem with modeling human behavior based upon theories of cognition is that cognitive models are difficult to quantify. Humans do not seem to make decisions based entirely upon mathematical formulae or deductive logic [75]. In contrast to standard machine-learning methodology, humans actually make poorer decisions when given too much information [45]. Instead, humans use heuristics, and these heuristics are not necessarily consistent between each individual, types of decisions, or even each time the same decision is made. They can be derived from individual preference, generalizations, culture, experience, or even learned from others [29].

Emotions also play a large part in human decision-making, as shown in neurological studies related to damage of the prefrontal cortex [6] and studies on the amygdala and Pavlovian responses [77]. There have been many attempts to incorporate emotion into agent decision-making, such as the Cathexis model [89], EBDI [47], and a Neuro-Fuzzy agent with emotional intelligence [78]. However, as shown by Martínez-Miranda and Aldea, while emotion is important in human decision-making, emotional models tend to only perform well in the specific environment for which they were designed [58]. The concept of 'emotion' is not well-defined in psychology, making it also

difficult to quantify [80]. This makes a standard cognitive-based model for even a simple scenario almost impossible to accurately define and implement [72].

Time-sequence agent-based models, or agent-based models derived from sequences of observations over time, however, have been implemented with a great deal of success in a variety of areas other than human behavior. For instance, OptorSim uses predictive modeling to optimize the allocation of resources for file sharing, replication, and job execution in a Data Grid based on the sequence of jobs executed over time [7]. A model of Chilean agriculture designed by Berger predicted economic changes based upon irrigation and adoption of new farming technology [9]. LUCITA, or Land Use Changes In The Amazon, was developed to determine the effects of local farming on Amazonian deforestation [22]. Time-sequence traffic data for densely populated regions has been used to predict traffic forecasts in Germany, allowing travelers to select the most appropriate route to their destinations [91].

Likewise, single agents have been successful in learning patterns of human behavior from sequential observations. In an attempt to learn human behavior based on emotion in a smart-home setting, Leon et al. created the iSpace and iDorm, test facilities that used sensors to detect physical changes that are associated with certain emotions [56]. They used an analysis of sequential behavioral patterns along with Autoassociative Neural Networks that use physiological responses to predict the emotional state and likely behavior of the smart-home occupant. This allowed the agent to automate some of the smart-home's systems.

ILSA, an agent-based smart-home system designed by Guralnik and Haigh to assist the elderly, uses sensor readings to determine sequences of the occupant's behaviors [36]. It uses the sequential patterns of which sensors fire to determine which times certain activities take place, such as what time a person wakes up, and what time they go to sleep. The researchers that designed ILSA concluded that the order

of sensors firing over a time interval was important to learn the behavior patterns of the person living in the home.

MavHome, a smart-home designed by Cook et al., also uses sequential sensor readings for behavior prediction [19]. The goal of MavHome was to adapt to the behavior of its inhabitants by automating processes such as turning up the heat in the morning, or turning on the light and coffee maker after the bedroom alarm goes off. Instead of just using sensors for doors opening and closing, MavHome uses a wide variety of sensors, such as temperature sensors and sensors to monitor the lawn moisture level. Behavior patterns are learned online, and prediction algorithms are used to match patterns in order to determine which devices to operate within the home. Because the learning is online rather than offline, it uses a string compression algorithm, Active LeZi, to compress the behavior sequence and increase the agent's online learning speed. Active LeZi uses a variable order Markov model to predict the probability of the next behavior in the sequence, reducing computational time [32].

1.5 Sequentially-Based Models of Human Behavior in Multi-Agent Systems

Because of the relative ease of creating more believable results in video games, the behavior models of human NPC's, or non-player-character agents, is already frequently based upon sequential observations. In many First-Person Shooters, as well as in games like Forza Motorsport and Black and White, opposing agent actions mimic the actions of users playing the game. This provides the player with a challenge more suitable to his ability level and style of play, making it more enjoyable. It also makes the opposing agent's behavior seem more realistic without being too computationally difficult to calculate [88].

More recently, there has been some success in building believable agents in academic competitions. A framework has been developed for the RoboCup simulation

league using a combination of observed player behavior and case-based reasoning that makes training the robotic soccer player much more simple [54]. It learns behaviors from logs of human players' actions. It then turns these into cases to be used in case-based reasoning. These cases are then weighted automatically using a k-nearest neighbor classifier. This algorithm is run until the agent responds to a situation in a simulation in the same way as the previously stored, "test" agents. After the agent has learned a sufficient amount of behaviors from humans, it can then train other agents [23].

BotPrize is a competition using the game Unreal Tournament 2004 that uses the Gamebots system [1]. It is held every year, and is a DeathMatch First-Person Shooter type game. Human judges play the game along with the bots, and try to distinguish between human and agent players. The UT^2 bot from the Neural Networks Group at the University of Texas, Austin, originally used recorded human behavior sequences to navigate when the bot became stuck. This bot placed second in 2010 [50]. After the 2010 competition, the Neural Networks Group increased the role of human behavior imitation in the evolution of combat behaviors and in navigation [76]. Subsequently, they fooled more than 50% of the judges in the 2012 competition, tying for first place with another bot that also mirrored human behavior.

As agents based upon sequential observations become more believably human, it can clearly be seen that imitation is a valid way of representing human behavior. While it would be computationally difficult to model all possible behaviors in a simulation that is very open-ended, many simulations have a limited number of possible actions available to the agent, making imitation a good way of providing a believable initial decision process [87]. Automation of the actual calculation of transitional probabilities between behaviors would allow more time for the modeler to make adjustments as needed, making modeling easier, and providing a more accurate, believable end-product.

Chapter 2

Automatic Generation of Agent Behavior Models from Raw Observational Data

2.1 Introduction and Related Work

Agent-based modeling has been used to simulate traffic patterns, markets, supply chains, wildlife ecology, and networking. It is a popular method for simulating complex systems because of its ability to show emergent behaviors, or behaviors that arise from the interaction between the different agents. Unfortunately, the creation of an agent-based behavioral model can be a difficult task, especially when modeling humans that are involved in complex processes. Frequently, simulation models involving human decision processes are created using observed behavior sequences. This model development paradigm requires that both the programmer and the domain expert work together to create a computational model which correctly reflects the observed behavior.

One possible method of building a model of human behavior is by deep analysis of the human decision process and human cognition. This is, in essence, the goal of cognitive psychology, which tells us that the heuristics humans use to make decisions are highly varied and individualized [29]. Emotions also appear to play a large role in human decision-making, as shown in neurological studies related to damage of the prefrontal cortex [6] and studies on the amygdala and Pavlovian responses [77]. Attempts to incorporate emotion into agent decision-making, even though they have in some cases had limited success, have the tendency to produce models that only

perform well in the specific environment for which they have been designed [48, 58]. This is likely due, at least in part, to the difficulty in quantifying "emotion," a concept that is not clearly defined in psychology [80]. This lack of a clear model of the human decision-making process made an alternate method of deriving a decision process an attractive alternative.

Agent decision processes that have instead been derived from sequences of behavior observed over time have proved successful in many areas, including human behavior modeling. For example, in smart home studies, sensor pattern readings have been used to determine human behavior patterns in order to automate heating and lighting systems in accordance with the owners' lifestyle [19, 36, 56]. In the RoboCup competition, a framework was developed not only to learn from logged human behavior, but to then train other agents by using the behavior it had learned [23]. The 2012 BotPrize competition, an Unreal Tournament Death Match-style game where human judges attempt to distinguish between AI-players and humans, had a tie for first place between two bots that used mirrored human behavior sequences, fooling more than 50 percent of the judges in the competition [76]. Thus, there is ample evidence in different settings that agents that effectively and believably simulate human behavior can be built by deriving decision processes from observed sequences of human behavior.

In robotic planning, there has also been some success in deriving decision processes from observed behavior. The learning of primitives [8] or low-level actions [35] using variations of HMM's enables robots to learn by imitating behavior, although these methods necessitate online rather than offline learning. More recently, a method has been proposed to enable robots to learn offline using human-readable text files [49]. This method, however, requires natural language processing and the careful construction of an appropriate ontology, unlike our research, in which the task names are provided by domain experts, and behavior is recorded by trained observers.

2.2 Problem Description

In simulation, agents have required the specialized skills of AI experts working together with domain experts to create the needed agent behaviors. In contrast, our research aims to develop algorithms and tools to automatically build these agents' behaviors using the raw observational data. That is, we want to take observed workflow data as input and output a generalized behavior model. Also, since these models will almost certainly require some modification, we propose to develop tools for domain experts, who are not AI experts or developers, to be able to modify these behaviors as needed.

In this chapter, we present MAGIC (Models Automatically Generated from Information Collected), an algorithm for extracting behavior models from raw observational data consisting of time-stamped sequential observations of the subject's behavior [70]. Our behavior model, described in Section 2.3.1, resembles a Markov Decision Process (MDP), but with added support for cyclic behavior and additional nodes known as **decision points** that indicate when the agent requires outside input in order to proceed. In order to demonstrate the ease of modification of the behavior model for use in simulation, we have also developed an editing tool that allows the model to be altered, and illustrated its use in a 3D simulation of a nurse administering medications to patients on a hospital floor.

2.3 Research Approach

We test our algorithm both in a synthetic test setting and a hospital setting where we build a simulation of a nurse as she carries out a medication administration process in a hospital. Data for the nursing simulation was gathered by following several nurses for 6 weeks as they administered medications to their patients [43, 82]. We then present our real-world simulation to a focus group consisting of domain experts. Our

experimental results demonstrate that the MAGIC algorithm can automatically build appropriate behavioral models.

2.3.1 Behavioral Model: Sequential Compressed Markov Decision Process

The type of behaviors we wish to model can almost be captured using a Markov Decision Process [68]. However, since MDP's do not allow for an internal state, they cannot be used to represent a finite loop of a length prescribed by an outside input. For example, in our healthcare domain, we need to represent the fact that a nurse will administer a fixed number of medications to a patient, so she will repeat a finite set of tasks some fixed number of times, such as 5 steps for each one of the 3 medications. We need a behavior model that can also represent these repeated sequences.

In this study, we created a variation of the Markov Decision Process that we will refer to as a Sequential Compressed Markov Decision Process, or **SCMDP**. This SCMDP extends the basic MDP by including decision points which have direct links to other states based on external inputs instead of a transition probability.

Definition 2.1. (Sequential Compressed Markov Decision Process) An SCMDP consists of an initial state s_0 and an end state s_n both taken from a set S of states where $|S| = n$, a transition function $T(s, p, s')$, a set of decision points $D \subset S$, and a set of decision point transitions $P(d, s, e)$ where $d \in D$ and e is some external input.

In the SCMDP, states correspond to tasks performed by the agent, such as "wash hands" or "enter room." The transition function T gives the probability p that the agent will transition from one state to another, therefore doing the corresponding task. All transition probabilities from any given state will always add to 1, as they do in an MDP. Start and end states s_0 and s_n are designated to account for the fact that only certain tasks are likely to occur at the beginning or end of a sequence.

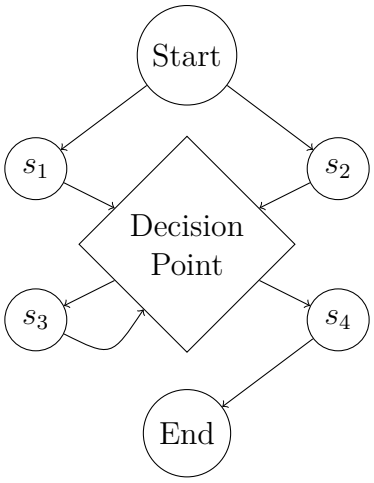


Figure 2.1 Example SCMDP.

The decision points D are a set of special states within the decision process. They represent the entrance to a cycle. Each decision point has at least two edges extending out from it. One edge goes to the first state in the cycle, and the other to the action that is to be taken after the cycle ends. The cycle begins and ends due to some external information e. The transitions out of decision points are represented by P. For example, a nurse agent might repeat the same set of tasks for each medication that must be administered to a patient. The external information in this case is the number of medications that the patient requires. The decision point keeps track of how many medications have been administered thus far and ends the cycle when there are no more medications to administer. It is possible to have more than one transition out of a decision point, therefore requiring more than one piece of external information, such as whether the medication the patient needs is available, and whether it is located in the medication room or the pharmacy.

Figure 2.1 shows a simple example of an SCMDP. Note that at the decision point, the agent can either repeat the cycle by going back to s_3 or end the sequence by choosing to go to s_4. In this example, the cycle consists of only one state, s_3, but

there could be any number of states before the agent gets back to the decision point. The decision between s_3 and s_4 is made using external information not shown in the diagram. In practice, this external information will depend upon the domain that the SCMDP is modeling.

2.3.2 The MAGIC Algorithm

The MAGIC algorithm, shown in Figure 2.2, takes as input a text file of sequential task observations and outputs an SCMDP. This input text file consists of a sequence of observations O, where each observation $o \in O$ is a sequence of tasks, $o_i = (t_1, t_2, \ldots t_{ki})$, that we have observed a person perform. For example, one observation corresponds to the sequence of tasks that we watched a nurse perform from the time she entered a patient's room on Monday 9:32 am until the time that she left the room. We assume that all of the observations have recognizable start and end points. In the nursing example, these start and end states correspond to a change in the patient's room number.

The MAGIC algorithm tries to identify and extract cycles in the raw input data, which is especially difficult given the fact that the data might contain errors in the form of transposed tasks. For example, in the observation $t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_3, t_5$, t_4, t_8 the set of tasks t_3, t_4, t_5 should be recognized as a cycle because they appear twice, even if in different order: the first time as t_3, t_4, t_5 and the second time as t_3, t_5, t_4. This match is performed by the CONTAINS procedure, shown in Figure 2.2, which tells us if the list of observations o contains the set of tasks t anywhere within it, contiguously, and then returns the indexes i, j within o that mark the start and end of the set of tasks t, or NIL if they are not contained in o.

The MAGIC-ASSISTANT procedure takes as input a single observation o, the list of tasks S that have been replaced by a cycle, the current list of cycles found C, and an integer m which is the maximum number of tasks that we will allow in a cycle.

```
MAGIC (O)
1  C = [ ] // List of cycles
2  m = k // Maximum number of tasks in a cycle, defined by user
3  O' = [] // Updated List of Observations
4  S = [] // List of lists of tasks that have been replaced with cycle pointers
5  for o ∈ O
6        o, S, C = MAGIC-ASSISTANT(o, S, m, C)
7        O'.append(o)
8  T = CALCULATE-TRANSITIONS(O', S)
9  return T, S

CONTAINS(o, s)
1  if ∃_{0≤i≤j≤|o|} o[i..j] ∩ s! = ∅ // Does o contain s, sequentially?
2        return i, j // If so, return the start and end points in o.
3  return NIL

MAGIC-ASSISTANT(o, S, m, C)
 1  t = ∅ // List of repeated tasks
 2  for c ∈ C
 3        for j = 0 to |o| − (|c| + 1)
 4              s, e = CONTAINS(o, c)
 5              if s, e ≠ ∅
 6                    o[s..e] = c // Replace the list of tasks with a pointer to the
                      cycle in the cycle list
 7                    S.append(o[s..e]) // Add list of repeated tasks to list for
                      transition calculations
 8  while m > 2
 9        for i = 0 to |o| − (m + 1)
10              j = i + m − 1
11              t = o[i..j] // Set of contiguous tasks taken from the observation
12              s, e = CONTAINS(o, t)
13              if s, e ≠ ∅
14                    if ¬∃_{c∈C} t ⊆ c // If t is not a subset of an old cycle
15                          C.append(t)
16                    o[s..e] = t // Replace the list of tasks with a pointer to the
                      appropriate cycle
17                    S.append(o[s..e]) // Add list of repeated tasks to list for
                      transition calculations
18        m = m − 1
19  return o, S, C
```

Figure 2.2 The MAGIC algorithm. The CONTAINS procedure tells us if the list of observations o contains the set of tasks t anywhere within it, but contiguously. The MAGIC-ASSISTANT procedure identifies cycles, checks if they are subsets of existing cycles, and records any new cycles found.

MAGIC-ASSISTANT first checks to see if o contains any cycles that are already in the cycle list C, as seen in lines 6–7. If any are found, then it modifies o so that tasks that we recognized as belonging to c are replaced with a pointer to c in C (see line 6). The list of tasks that have been replaced is appended to S, so that the transition probabilities within the cycle can also be calculated.

MAGIC-ASSISTANT then steps through the observation sequence (see lines 8–18) selecting the maximum number of tasks in a cycle, converting them to a set s where $s = o[i..|s| - 1]$ and using the CONTAINS helper function to check for repetitions of that set of tasks in the same observation, that is, checking for a cycle. If a new cycle is found, we determine if it is a subset of one of the cycles that is already on the cycle list C. If the cycle is not yet on the list, it is added to C. The cycle is then replaced in the observation o with a reference to its location on the cycle list C. Finally, MAGIC-ASSISTANT returns the new modified observation o and the list of tasks S that have been replaced by a cycle $c \in C$.

The MAGIC procedure repeatedly calls MAGIC-ASSISTANT for each observation o and appends the new modified observations to O'. Finally, it calculates the transition probabilites T using the new O' and the list S by adding how many times a state follows another one and using the proportions as probabilities. In other words, if state s_6 appears right after s_2 in $1/3$ of the observations where we see s_2, then we set $T(s_2, 1/3, s_6)$.

As an illustration of the way that the MAGIC algorithm functions, consider the set of observations in Figure 2.3, which simulates the attempt to play fetch with a dog who doesn't seem to understand the concept of giving the ball back. Since the length of the maximum observation is 8, we know the longest possible cycle will be 3, because the start and end states cannot be in a cycle. Thus, we set $m = 3$ in MAGIC. However, there are no cycles 3 tasks in length. The first and only cycle found is $c = (\textit{throw-ball}, \textit{chase-dog})$, which also matches the set $(\textit{chase-dog}, \textit{throw-ball}.)$ Each

go-out, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *go-in*
go-out, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *go-in*
go-out, *throw-ball*, *go-in*
go-out, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *go-in*
go-out, *throw-ball*, *chase-dog*, *chase-dog*, *go-in*
go-out, *chase-dog*, *throw-ball*, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *go-in*
go-out, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *throw-ball*, *go-in*
go-out, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *throw-ball*, *chase-dog*, *go-in*

Figure 2.3 Example input data for MAGIC algorithm.

time c is found, the list of tasks that are replaced by the pointer to c in the list of cycles C is added to the list of lists of tasks S, to be used in transition calculations inside of the cycle. An illustration of the SCMDP produced by MAGIC is shown in Figure 2.4. At the decision point the agent needs the external knowledge of whether or not it has the ball, and whether or not the dog wants to play. If the agent has the ball, it can throw the ball. If not, it must chase the dog to get the ball. If the dog doesn't want to play any longer, the agent will go inside. Going outside is always the first event in the sequence, and going back inside is always the last event.

The cycle created by our decision point ensures that, after completing the tasks of throwing the ball and chasing the dog, the agent returns to the decision point to once again make a decision based upon who has the ball, and whether or not the dog wants to play. This allows behavior that is based upon the human behavior pattern, but does not necessarily repeat one particular logged observation. For instance, if the agent goes outside and the dog does not want to play, the agent will go inside again. Likewise, the agent would continue playing fetch with the dog for more than three cycles if the dog still wants to play.

The modeler and the domain expert must choose the specific external inputs needed at the decision nodes. In this simple case, it is easy to determine that the input is simply whether or not the dog wants to play. In the case of a more complex

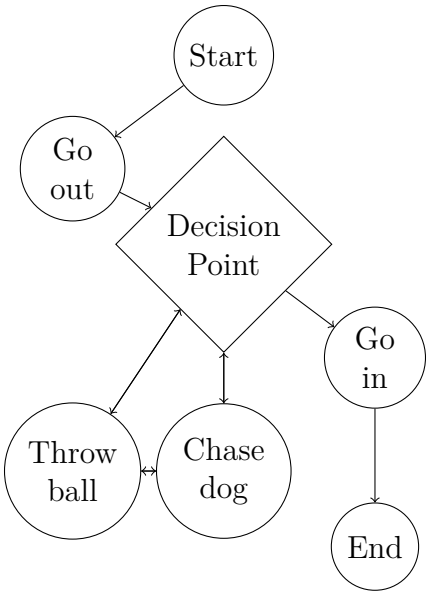


Figure 2.4 Example SCMDP produced using MAGIC algorithm

model, however, the domain expert may need to tell the modeler what the agent would need to know in order to proceed. Well-named tasks in the logged data make this process simpler, so it is important for trained observers who are logging behavior to be as accurate and clear as possible in naming tasks. It is likewise important that they remain consistent. If the same task is given two different names by observers, it will appear as different tasks in the final model.

2.4 Testing

To verify both the reliability and the believability of the agent behaviors produced using the MAGIC algorithm, we performed two separate tests. We tested the reliability of the algorithm using synthetic data. After determining the algorithm's reliability, we tested the algorithm's validity using real-world data by creating a simulation that we subsequently showed to a focus group of domain experts.

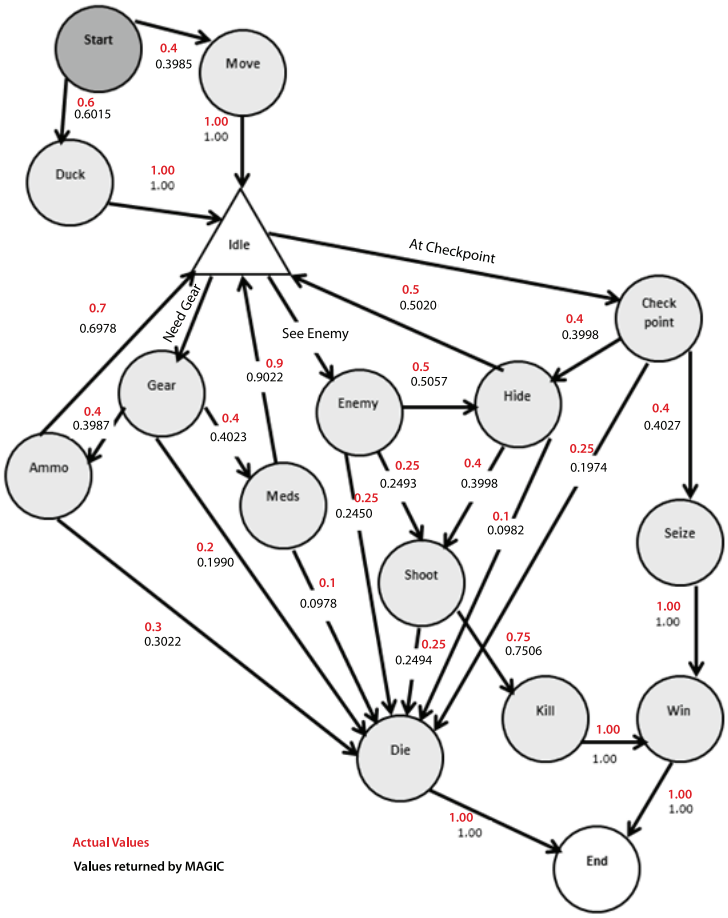


Figure 2.5 SCMDP used for testing the MAGIC algorithm. The numbers in red (above) are the original transition probabilities in the SCMDP. The numbers in black (below) are the probabilities found by MAGIC.

2.4.1 Validation Using Synthetic Data

In order to test how well MAGIC can extract cycles from raw data, we performed a test in which we created a synthetic model of a simple agent from which we could generate observational sequences. We then used the MAGIC algorithm to attempt to recover the original model from the observations.

The SCMDP we created mimics a player's movements in a first-person shooter "capture the flag" game. The agent has a single decision point called *Idle*. At this point, the agent needs to know if it is injured, needs ammunition, sees its opponent,

or is at the checkpoint that must be seized in order to win the game. There are two possible initial actions: crouch or duck, and there are two possible final stages: win or die. The SCMDP used for this test is shown in Figure 2.5.

We used a Python script to generate 10,000 strings from the SCMDP and fed these as input 10 times to the MAGIC algorithm, for a total of 100,000 randomly generated strings. The resulting SCMDP mirrored the original's pattern, providing an appropriate decision graph for an agent in a first-person shooter "capture the flag" game. The transition probabilities found by MAGIC were, on average, within 0.19 percent of the ones in the original SCDMDP, with a variance of 0.08 percent, as shown by the black numbers (below) in Figure 2.5.

Our results show that, with the use of 10,000 strings, we are able to closely approximate the original pattern with minimal deviation between individual test runs. The low error rate in transition values indicated that, by adding enough data, we were able to overcome the disadvantage of unusual behavior patterns, allowing us to recover the correct pattern of behavior using the MAGIC algorithm. The identification of task cycles enabled us to determine the location of the decision point, indicating that the *Idle* state is a state where the agent would require further information before making a decision, rather than simply relying upon a percentage chance of a transition.

We then performed further tests on this SCMDP by adding Gaussian white noise with 1% variance to the input data, meant to simulate the type of errors we might encounter in data gathering and subject observation. The addition of this noise did not disrupt the location of the identification of the decision point. It did cause a minimal error in transition values, which was easily correctable by removing transitions that had less than one percent chance of occuring. This slight adjustment to transition calculations also enabled better compensation for occasional unusual behavior patterns.

Table 2.1 Example of nursing data.

Room Number	Behavior
628	enter_room
628	greet_patient
628	scan_patient_id
628	review_patient_computer_record
628	review_patient_med_box
628	scan_patient_meds
628	document_med_admin
628	scan_patient_meds
628	scan_patient_meds
628	document_med_admin
628	scan_patient_meds
628	document_med_admin
628	review_patient_med_box
628	scan_patient_meds
628	prepare_meds_for_admin
628	administer_meds
628	prepare_meds_for_admin
628	setup_for_med_admin
628	administer_meds
628	other_care

2.4.2 Validation With Real-World Data

A pilot study of the nurse medication administration process was conducted in a hospital setting [42, 43, 82]. In this study, over a 6 week period of time, nurses were shadowed by trained observers, and their activities were recorded using an iPad application. The actions used by the observers were chosen by domain experts. Observation data from 6 of the 17 observed nurses were used for the study, and the resulting files were combined into a CSV file. The start and end of each observation sequence was determined by when a nurse entered and exited a room, as evidenced by the room number in the log files. In total, there were 10,391 tasks recorded which together comprised 313 observations.

An example of a subset of the data used is shown in Table 2.1.

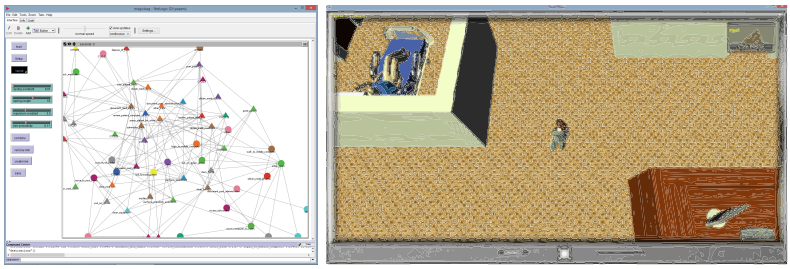


Figure 2.6 The MAGICBAG Tool (left) and the NurseView simulation (right). Simulation video at `http://youtu.be/JH94PolDhZQ`

Despite the limited amount of sample data, we were able to achieve some success using the MAGIC algorithm. We were able to identify 12 decision points needing external information, such as the number of medications the patient required, or whether or not the patient needed special medication. Some of these were less obvious in nature, such as whether or not the nurse needed to wear gloves, whether or not the patient needed the medication explained, or whether the patient refused to take the medication.

The nursing study was particularly interesting because the nurses had two distinct approaches to patient care, as identified by domain experts (clinicians, in this case). We have referred to these approaches as bundled and unbundled. Nurses that took the unbundled approach visited a patient's room to administer medication, and then returned later to perform any other necessary tasks, while nurses that took the bundled approach performed all required tasks during the same visit. The SCMDP we obtained from the test data reflected the fact that it contained both methods, as indicated by the decision point that requires knowledge of whether or not the patient requires other care than simply administering medications. While this pilot data set provided us with the location of the appropriate decision points, because of the difference in approaches to patient care, and the number of possible actions taken, it will be necessary to have a greater number of observations to ensure the correct transition values.

Despite the smaller size of the data set, by using this format, we were able to create simulations using both NetLogo and the Unity3D game engine that can read the text file and use it as a logic controller for the nurse agent's behavior, as seen in Figure 2.6. This allowed the nurse domain experts to visualize current medication administration processes.

2.4.3 The MAGIC-BAG and BACKPACK Tools

Although the output we recieved after using the MAGIC algorithm was human-readable, we wanted a more intuitive way for modelers to edit the decision process. We therefore designed the MAGICBAG (MAGIC Behavior Adjustment Graph) Tool to enable modelers to more easily adjust agents' decision graphs without requiring them to have any programming knowledge. The MAGICBAG Tool, shown in Figure 2.7 allows modelers to add and remove links between states, combine states, remove states, and save the results so that it can be used in the same way as the initial file. The tool automatically updates transition values as links and states are changed, normalizing them so that the probabilities for the other links remain constant and the total transition probabilities from each state always sum to 1.

Although the MAGICBAG Tool enables modelers to duplicate existing states, it does not allow them to create new states. As the states are actions that will be taken by the agent, adding a state with an action that has not previously been defined would make the resulting decision process unusable by the current simulation.

The NurseView BACKPACK, or Behavioral Agent Color-Keyed Pictorial Automated Controller Kit Tool, shown in Figure 2.8, is a variation of the MAGICBAG Tool. This variation was designed to work with the NetLogo and Unity simulations used in the medical administration research study. It contains additional layout information for the graph, making it simpler to view and alter.

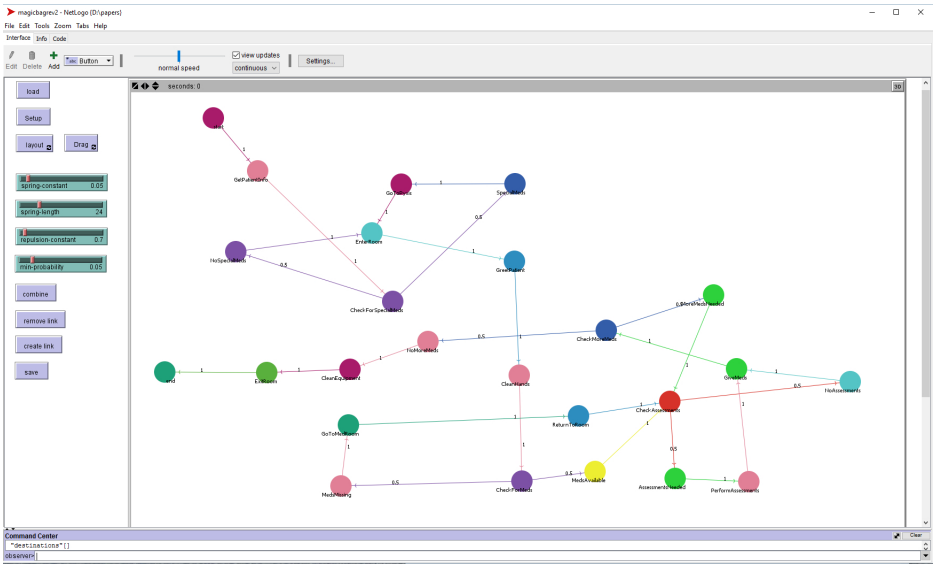


Figure 2.7 MAGICBAG screenshot

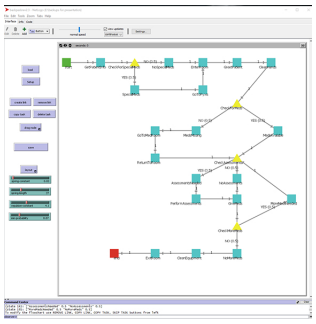


Figure 2.8 BACKPACK screenshot

2.4.4 Believability

In a meeting with the clinicians, they indicated that the nurse agent's behavior appeared believable. When asked by a member of our research group whether the nurse actually had to walk to the med room repeatedly, as our simulation showed, the clinicians stated that she did, and indicated that they would be interested, based on the visualization, in having the nurses wear pedometers to find out how far they had to walk during their shifts.

It is interesting to note that the simulated behavior appeared believable to the clinicians, who, as health care professionals, were similar to the nurse being modeled [42]. To those who were not health care professionals, there were things about the nurse agent's behavior that appeared strange, making it less believable to them. This supports the theory that believability is highly dependent on the observer's similarity to the human being modeled. We will discuss this further in Chapter 3.

2.5 Summary

As cognitive modeling is difficult, imitation is a viable alternative to achieve believable human behavior in simulation. Statistical analysis of observed data allows us to achieve a pattern of human actions, essentially simulating human behavior by mimicking human behavior.

While building behavior models by hand can be complex and time-consuming, there is a better alternative. We have shown that it is possible to derive an agent decision process using the MAGIC algorithm which encapsulates the observational data in a small behavior model (SCMDP) that responds to external input, provided there is sufficient data, and tasks are labelled consistently.

Even with an automatically generated decision process, it will be necessary for an expert in the area that is being modeled to review the results. The process, however, will be less complex and time-consuming than making all of the necessary calculations by hand. The simple, standardized output format used in this study is easy to parse, allowing adjustments to be made quickly, and making it easy to load in a wide variety of simulation environments.

The meeting with the clinicians was indicative of one of the limitations of believability in a simulation environment. A member of our research team, who was not a healthcare domain expert, did not find it believable that a nurse would walk repeatedly between the patient's room and the med room. The clinicians, however,

did find it believable. This supports the notion that believability depends not only on the behavior of the agent, but also upon the target audience. In Chapter 3, we observe the difference this makes in test results when a video of an agent is observed by two different groups of people, one that is similar to the people whose actions were used to create the agent, and one that is dissimilar.

Chapter 3

Generation of Believable Agents Using FPS Player Data

3.1 Introduction and Related Work

The nursing simulation, shown in Chapter 2, involves behavior patterns in a static environment. There are many types of simulations, however, which involve changing environments. These types of simulations require additional limitations in order to be sure that the actions taken by the agent are appropriate to the current state of the simulation environment. Video games provide a good example of simulations with changing environments, providing an additional platform in which to test the MAGIC algorithm.

Recently, there have been many crossovers between the use of artificial intelligence in the video game industry and its use for academic research. Improvements in graphics have enabled developers to devote more resources to artificial intelligence, allowing the use of more complicated algorithms in gaming. At the same time, the necessity of quick decision-making in games has driven developers to optimize existing algorithms, enhancing their performance. This has improved agent planning, which benefits both the gaming industry and research.

Video game developers must solve a large number of complex problems in a relatively short period of time, and with limited resources. Until recently, the necessity of using the majority of a computer's resources simply to render graphics limited developers' ablility to focus on AI [53]. High-end gaming studios used algorithms that

were developed in the 1960's and 1970's for pathfinding and decision-making in order to focus on graphical performance. With the recent increase in computational power that is available to them, it has become possible to use more complex algorithms to solve these problems [44]. However, as game worlds increase in size, the complexity of the problems has also increased. It is no longer possible to use story script-based AI behaviors for each agent encountered in the game world. Game performance and realistic behavior must be balanced in order for the game to function properly. These developments have created an environment in which behavior algorithm optimization is of great importance. As such, game developers have implemented more recent algorithms designed by researchers in academia, but they have also designed new procedures to optimize these algorithms to run in large-scale environments at lower computational costs [60].

Due to their emulation of real-world environments, game engines have become popular tools for simulation. The built-in collision detection, physics simulation, and graphics enable developers to concentrate on the environment they are attempting to simulate, rather than having to create everything from scratch. Game engines are used to simulate things like recreations of historic structures [74], medical training environments [33], and disaster recovery [85]. This makes a game-like environment a good test bed for AI development, and allows researchers to build on the tools developed for gaming.

3.2 Problem Description

While we have been able to partially automate the model generation process in a single simulation using the MAGIC algorithm, introduced in Chapter 2, we would like to show that the procedure works in a variety of simulations, increase the speed and accuracy of believably human decision-making by taking into account changes in the simulation environment. Given an appropriate simulation, we would also like to

show that it is possible to further automate the model generation process by recording observations programmatically, rather than by using trained observers.

The problem that we have discovered, as we will see in Section 3.3, is that the inclusion of additional information into an existing model increases that model's complexity. Adding more terms allows for more error in the model. This means that we need to find a balance between the amount of information included in the model, and the complexity of the model. In this chapter, we compare models built using our original process versus models built using the same process with the addition of a rule set to limit the agent's actions based upon its current environment.

3.3 Research Approach

We build a test environment in the form of a video game that is hosted online. The video game differs greatly from the nursing simulation, allowing us to establish whether the MAGIC algorithm will still function in different types of simulations. Since the game provides feedback to the player, we are able to take that feedback into account when attempting to model a player, and increase the accuracy of decision-making by narrowing the context in which the decision occurs. The game records actions as they are taken, eliminating the need for human observers. Automated recording provides consistency of observations, reducing error in measurements. It also eliminates latency between the time the action is taken and the time it is recorded.

It is important to distinguish the fact that we are attempting to simulate the human player, rather than the actual game character. Unlike the nurses in the previous example, the player has a certain amount of knowledge about the condition of the game character. In the nursing simulation, the nurse had to obtain all of the information necessary for decision-making from outside sources, such as the patient, or the patients' health record. In the game, however, there is knowledge, such as whether or not the character is injured, that is clearly available to the player. It is

possible to take this knowledge into account when making decisions. However, the player does not have complete knowledge about the state of the game environment. For example, he does not have exact knowledge about his character's maximum speed or jump height, and he does not know the location of all of the enemy characters, or how they will react to his actions. For the purpose of this chapter, we will refer to the player's knowledge as **internal knowledge**, since it is knowledge that is not received from another agent or from the environment.

In the video game, we have more information than was available in the nursing data set. Due to privacy regulations and safety concerns, we could not track the nurses' exact positions, requiring us to rely on general information about the nurses' locations when they performed various tasks. In this case, we have the exact location of the player's character at the time each action is taken. Precise locations enable us to find a player's most likely path though the game world as well as the actions he is taking at the time. The character's positional information is also useful in determining whether or not the player is in an unsafe location.

The other important difference between the nursing simulation and the video game is that the nurses have a set routine that they follow, and the game players do not. There may be some variance in a nurse's routine, but there is no guarantee that game players will do anything in a set order. They have the same general objectives, the same game environment, and the same set of available actions, but their playstyles may vary.

The game, NoMadMan, is a first-person shooter-style video game written using the Unity3D game engine that is designed to capture user data in order to create a player bot. NoMadMan is browser-based, making it accessible to players on multiple platforms from any location that can access the website. The game has a story-based introduction, followed by a tutorial level that teaches the users how to play the game. After the tutorial is completed, players are automatically sent to the test level. In the

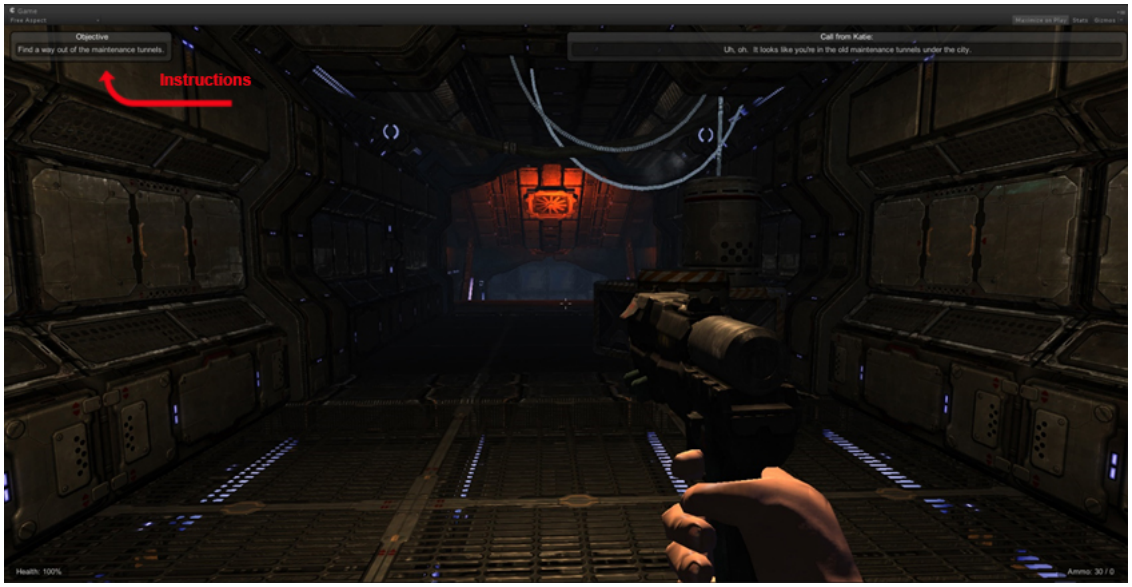


Figure 3.1 Example game screen

test level, users attempt to reach a specific location without being killed by hostile computer-controlled characters. Users are given instructions as to how to proceed as the level progresses, as shown in Figure 3.1.

The players can move at different speeds and jump as well as fight, and they are able to interact with certain objects in the environment. The cursor changes when the player's targeting reticle is over those objects in order to indicate that they are usable objects. Players can also pick up extra ammunition by walking over it. The ammunition packs are highlighted in order to make them more obvious to the player. Players regenerate health when they are not in combat. If a player's health falls to zero, the player dies and is reset to the beginning of the test level.

While users play the game, their movements are stored in a string that is subsequently written to a text file housed on the game server. Because this string is automatically generated by the game code, data is interpreted programmatically, rather than by a human domain expert, as was necessary in the nursing simulation. Each file is stored separately, named using a random number, and the player's name,

Table 3.1 NoMadMan SCMDP conditions

Injured	Has Ammo
True	True
True	False
False	True
False	False

if the player chooses to submit it. The player's name is removed prior to calculation in order to keep the data anonymous.

During the test level, each player's movements, current position, knowledge, and current time are recorded. The player commands recorded are Move, Run, Interact, Jump, Attack, Crouch, Reload, and Get Ammo. The player's position is recorded in x, y, z format, with y being the vertical axis. To attempt to record what the player knows, some additional information is recorded. In this case, we have assumed the player's internal knowledge includes two boolean values: whether or not the player is at lower than half health, and whether or not the player has less than 10 units of ammunition remaining. As the position of enemy characters is known, whether or not the player is in combat can be determined by a combination of the player's location, actions, and current health. Time is recorded in seconds from the time the level was loaded. After the player completes the level, the string is automatically uploaded to the server as a text file in comma-delimited format.

In NoMadMan, there are two conditions that are made obvious to the player through the game interface - whether or not the player is injured, and how much ammunition the player currently possesses. If we use these conditions to define separate SCMDP's, we have four, as shown in Table 3.1.

Ignoring our assumption about the player's internal knowledge about health and ammo, and instead basing the agent's choices on the player's internal knowledge about

a change in the simulation environment, we have only two SCMDP's. One, which we will refer to as **Hazardous**, is used when the player character is being attacked by an enemy character. The other, **Non-Hazardous**, is used when the player is in a safe location. Although there is some variance in player actions, this makes it simpler to make decisions that are appropriate to the current environment. For instance, if there is no enemy present, it does not make sense for the player to run around attacking at random. If there is an enemy present, it does not make sense for the player simply to run in circles while jumping.

By giving context to the decision-making process, it is possible to make more appropriate decisions [69]. Each time a condition is changed, it is determined which SCMDP should be used. The corresponding state is found in that SCMDP, and then the next action is taken.

3.3.1 Conditions and MAGIC

The decision process uses a hierarchy of rules based upon the player's internal knowledge to determine the appropriate SCMDP for the current environment. It continues through the currently loaded SCMDP until an *end* state is reached or until one of the boolean values is changed. If a boolean value is changed before an *end* state is reached, the agent goes to the *start* state of the new SCMDP to determine the next appropriate action. The decision process is iterative, and proceeds through the following steps:

1. Load the default SCMDP.
2. Take an action.
3. Check rules for a change in environment.
4. If there is a change in environment, load the SCMDP for the new environment.

a) Go to *start* state in the new SCMDP.

b) Return to 2.

5. If *end* has been reached, exit. Otherwise, return to 2.

Location information is extremely important when simulating human behavior. Certain actions taken by humans only make sense in a certain, specific location. Others are tied to a more general location, such as those previously noted in Section 3. Lack of information data makes the creation of a believable human model extremely difficult, as was the case in the Nursing study mentioned in Chapter 2. Due to HIPAA regulations, observers were not permitted to enter rooms with the nurses, so they were not able to record accurate measurements of the nurses' locations. Instead, they were only able to log general locations, such as the room number, or "Hallway."

In our game simulation, actions such as "Attack", "Interact", and "Get Ammo" are not appropriate in certain locations. The "Get Ammo" action is automatically taken when a player is in a certain location, while "Interact" is only possible in certain locations. "Attack" is possible in any location, but is not commonly used in locations where there is no enemy agent. Movement behavior is also important, as unbelievable movement patterns make the player agent appear less human.

As it is necessary for us to determine the player's location as well as the player's action, we find the players' most commonly used path using a Hidden Markov Model. We then create a grid of 1 unit squares, and store the path as a list of squares through which the player agent will move. Due to changes in speed and action, there will be a minor variance in path, but the overall path taken through the simulation will be similar no matter what actions are taken.

Due to the relatively small size of the sample set, our simulation was limited to a subset of the test level. This subset contained the entry room, which was a hallway containing objects that the player would collide with but could not interact with, and

the first room, which contained an enemy agent. The enemy agent was visible to the player, but would not attack until the player got close enough, meaning that it was possible for players to kill the enemy agent before they were attacked. The room also contains a number of objects that were not interactable, and some extra ammunition.

In the Non-Hazardous situation, we combined our location information with the action found using the MAGIC algorithm, so that the most common path was used for navigation, while the SCMDP was used for the current action. For instance, if the action to be taken was "Walk", the location to which the agent would walk would be the next location on the most common path. If the action was "Get Ammo", however, the player agent would move to the location with the ammunition before continuing along the most common path. In the Hazardous situation, however, player movement was less straightforward, as the players' paths were not consistent during combat. During combat, the players' movements changed accoding to the location of the enemy agent. While we used probabilities obtained from the observational data to determine the most likely direction of movement, we later found that there were more simple and effective ways to determine the player agent's path during combat. We explore this situation further in Chapter 5, where we present a potential simple rule to make bot movement during combat appear more believable.

While the MAGIC algorithm is used to create the SCMDP's necessary for the agent to make decisions prior to runtime, a conditional hierarchy is used to choose the appropriate SCMDP to use while the simulation is being executed. When using 4 SCMDP's, according to our initial hypothesis about the player's internal knowledge, the SCMDP would be chosen based upon whether or not the player has a low amount of ammunition and whether or not the player is injured. When using two SCMDP's, the SCMDP would be chosen based upon whether or not there is an enemy agent in the room. In either case, the simulation loads and stores all of the required SCMDP's in memory, then alternates between them each time a condition is changed. The actions

Table 3.2 Example of FPS game data.

Action and Location	Injury Status	Ammo Status	Elapsed Time
Move (6.9, 0.1, -4.2)	Not Injured	Has ammo	110.336
Move (6.9, 0.1, -4.2)	Not Injured	Has ammo	110.3535
Move (7.0, 0.1, -4.1)	Not Injured	Has ammo	110.3703
Move (7.0, 0.1, -4.1)	Not Injured	Has ammo	110.388
Move (7.1, 0.1, -4.1)	Not Injured	Has ammo	110.4046
Move (7.2, 0.1, -4.0)	Not Injured	Has ammo	110.4216
Move (7.2, 0.1, -4.0)	Not Injured	Has ammo	110.4391
Move (7.3, 0.1, -4.0)	Not Injured	Has ammo	110.4568
Move (7.3, 0.1, -4.0)	Not Injured	Has ammo	110.4764
Move (7.4, 0.1, -3.9)	Not Injured	Has ammo	110.4949
Move (7.5, 0.1, -3.9)	Not Injured	Has ammo	110.5117
Move (7.5, 0.1, -3.9)	Not Injured	Has ammo	110.5283

taken by the agent are determined by the correct SCMDP, and the direction of the agent's movement are simultanously determined by the most common path, with the exception of the Hazardous SCMDP, which also controls the agent's location.

3.4 Testing

A total of 135 files, each one corresponding to a different player's actions, were used to generate the agent player model. As data was recorded from the time the level was loaded, including the time before the player gained control of the character, all key commands entered before gameplay actually began were discarded. Duplicate entries that indicated that a player had not acted between frames were eliminated, and the time was adjusted accordingly. Blocks of data were separated according to the player's knowledge at the time of the action. An example of the data recorded by the program is shown in Table 3.2.

As shown in Table 3.2 above, even when the player was constantly moving, the data was captured at a high enough rate that there was some repetition in the player's

Table 3.3 Example of Processed Data

Action and Location	Injury Status	Ammo Status	Elapsed Time
start			
Move (35.8, 5.2, -4.3)	Not Injured	Low ammo	507.1673
Move (35.8, 5.1, -4.3)	Not Injured	Low ammo	507.2878
Move (35.8, 5.0, -4.3)	Not Injured	Low ammo	507.3371
Move (35.8, 4.9, -4.3)	Not Injured	Low ammo	507.3871
Move (35.8, 4.8, -4.3)	Not Injured	Low ammo	507.4207
Move (35.8, 4.7, -4.3)	Not Injured	Low ammo	507.4535
Move (35.8, 4.6, -4.3)	Not Injured	Low ammo	507.4871
Move (35.8, 4.5, -4.3)	Not Injured	Low ammo	507.5029
Move (35.8, 4.4, -4.3)	Not Injured	Low ammo	507.5363
Move (35.8, 4.3, -4.3)	Not Injured	Low ammo	507.5531
Move (35.8, 4.2, -4.3)	Not Injured	Low ammo	507.5691
Move (35.8, 4.1, -4.3)	Not Injured	Low ammo	507.5865

location between entries. This was easily solved by compressing those entries and adjusting the time accordingly. An example of the processed data is shown in Table 3.3.

Since the game was designed to be played within a web browser and was hosted online, it was likely that different players had computers with different capabilities, and that their internet connection speeds varied. We therefore used the original, unmodified data to determine the players' average framerate. We divided this by the average action speed, taken from the processed data. The average player speed was a fairly consistent rate of one action for every 1.97 frames, with a standard deviation of 0.74 frames. This gave us a believable action speed for the agent, since an agent that is not artificially limited by time is able to take multiple actions within a single frame.

In the data collected during gameplay, all keypresses are recorded. This means that "Move" is always recorded if the player is moving, even if the player is doing something else at the same time. For example, "Move" on one line followed by "Run" on the next line indicates that the player is running, since running requires

pressing both the key for directional movement and the shift key to run. In order to avoid confusion, "Move" without another simultaneous keypress has been changed to "Walk" in the included figures. There is also a potential pause between each action. However, since the probability of the next action is based upon the previous action and not the pause, "Pause" has been omitted in the included figures. "End of Path?" and "End" have been omitted for the same reason. In each case, the bot continues until the end of the path is reached, or until the bot is killed. In the latter case, it returns to Start. Transitions with probabilities of less than one percent have been omitted, as this was shown in Chapter 2 to be an effective way of accounting for error and outliers. Returns to start due to death have also been omitted, as this is not a player action. The remaining probabilities have been normalized.

After the data was processed, it was determined that there was not enough data to create 4 SCMDP's using our initial assumptions. Interestingly, the degree of character injury and the amount of ammunition possessed did not appear to affect the players' decisions. Instead, the players' actions were influenced by the change in the simulation environment. In the beginning of the game level, the player is in an empty hallway, without any items to obtain or enemies to attack. In the next room, however, there is both an enemy and a source of ammunition. The addition of an enemy greatly changed the players' actions upon entering the room. In the hallway, it was not possible to pick up ammunition, and attacking was not really an appropriate action to take. The lack of any opposition made actions involving movement much more likely than attacking. In the room, the players were more likely to move to avoid the enemy, or to attack.

The change in the players' actions appeared to indicate adaptation to the change in the simulation environment. This produced a **strong rule**, which is a rule that had a large impact on the actions taken by the player agent. In order to replicate the players' adaptation, we added a single boolean variable that represented whether or not the

environment was hazardous, used to represent the strong rule. The variable was changed based upon the player character's location in the game world, and whether or not there was an enemy present. This condition was used to create two new SCMDP's, the Non-Hazardous SCMDP shown in Figure 3.2, and the Hazardous SCMDP shown in Figure 3.3. These were combined in a hierarchy, with the rule-based decision at the top level.

The single SCMDP created without use of either strong or weak rules is very different, as seen in Figure 3.4. It does not take into account the player agent's location or status. This can result in a very erratic and much less believable agent.

While the change in environment produced a large change in behavior, our initial hypothesis regarding the player's internal knowledge did not. There were not enough players that remained injured long enough to determine if there would be a change in behavior due to injury, which rendered that data useless. Players also appeared to pick up ammunition whether or not they needed it, making that a **weak rule**, or a rule that did not have enough impact alone. As this effectively made no difference in the outcome, we disregarded this rule in favor of the strong rule regarding the existence of an enemy agent in the room.

3.4.1 Waypoints and Grid-Based Navigation

A **waypoint** is an agent's target location on a navigation path. The path is composed of a series of waypoints, and as the agent reaches each one, its next destination is set to the next waypoint in the list, until the final destination is reached. **Walkable** terrain is the only terrain the agent is permitted to traverse. Walkable terrain is used for the agent's navigation, and can be viewed as a grid. This form of navigation is also known as **Grid-Based Navigation** [34]. In the example shown in Figure 3.5, the red squares are waypoints numbered in sequence. The agent's path would be determined by first navigating from waypoint 1 to waypoint 2, then from waypoint 2

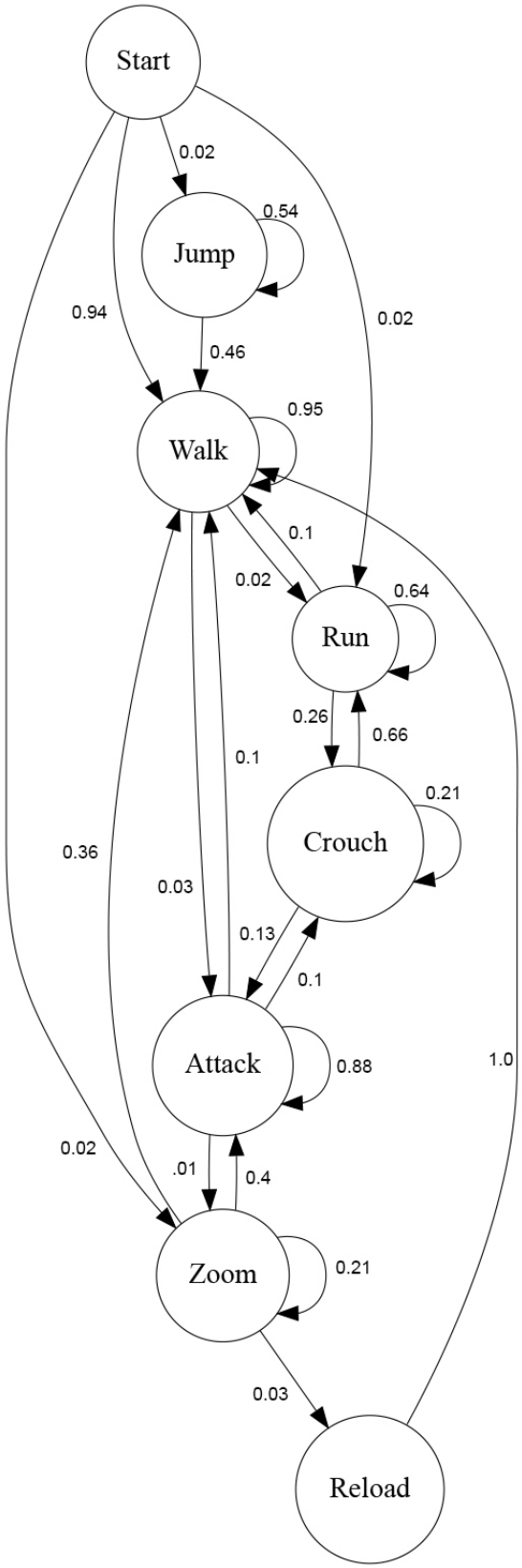


Figure 3.2 Non-Hazardous SCMDP

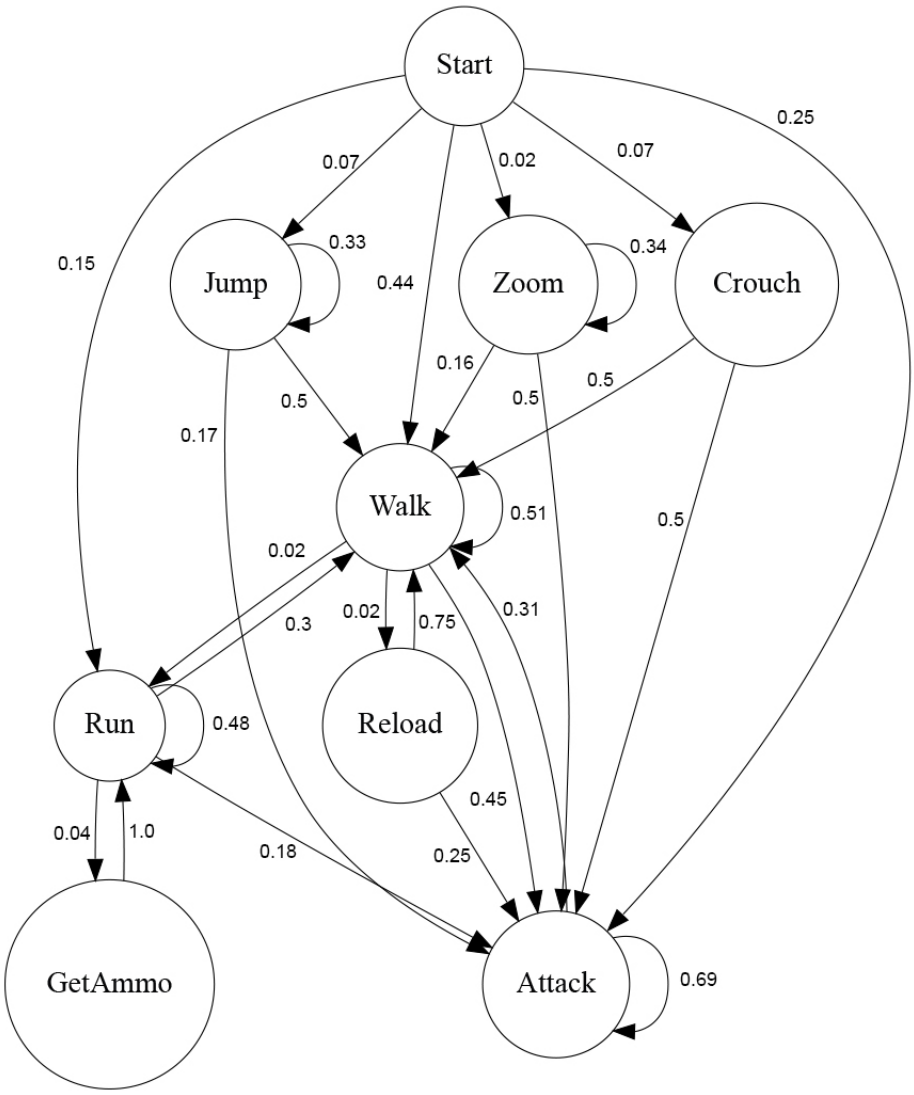


Figure 3.3 Hazardous SCMDP

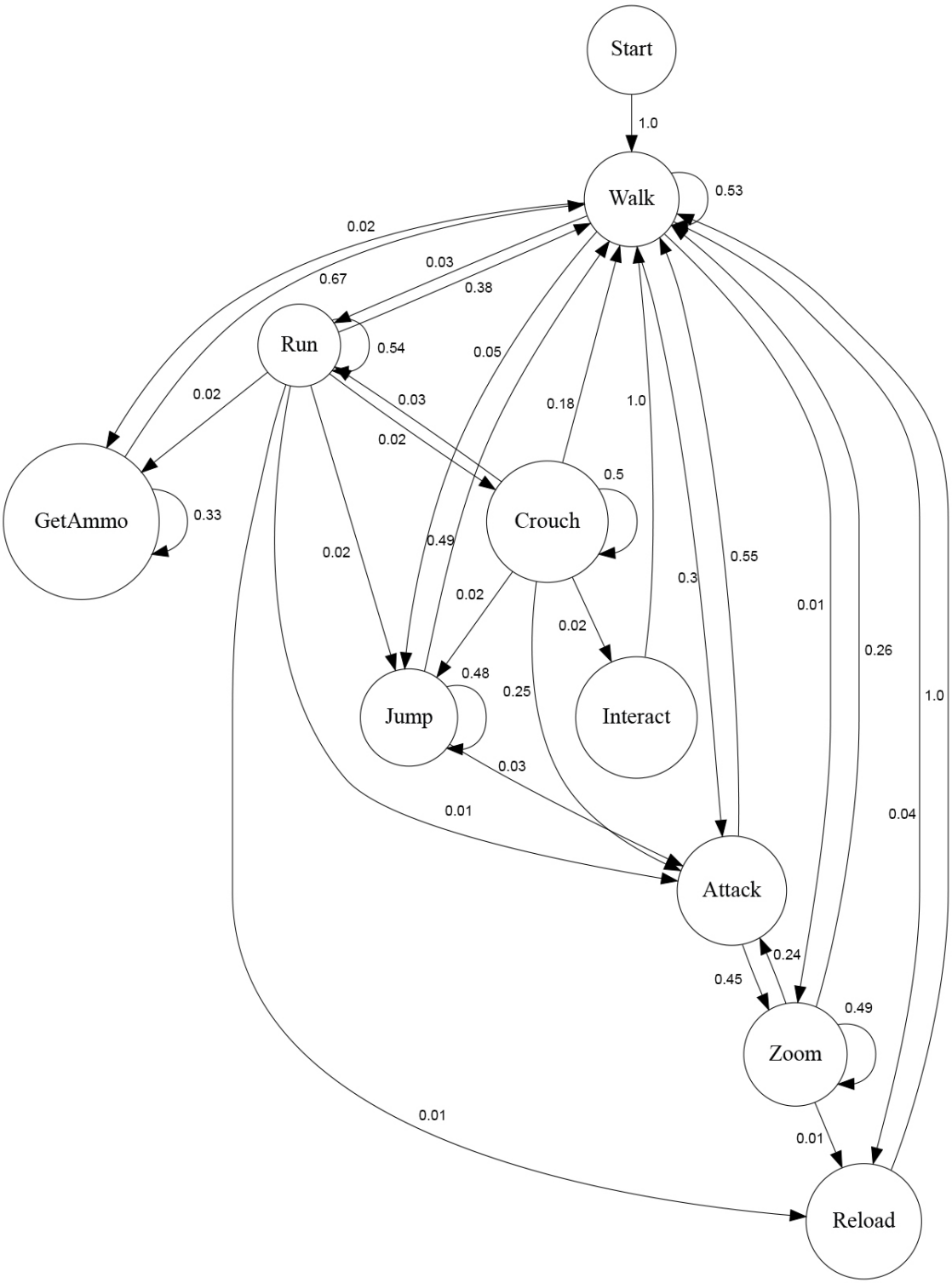


Figure 3.4 Single SCMDP

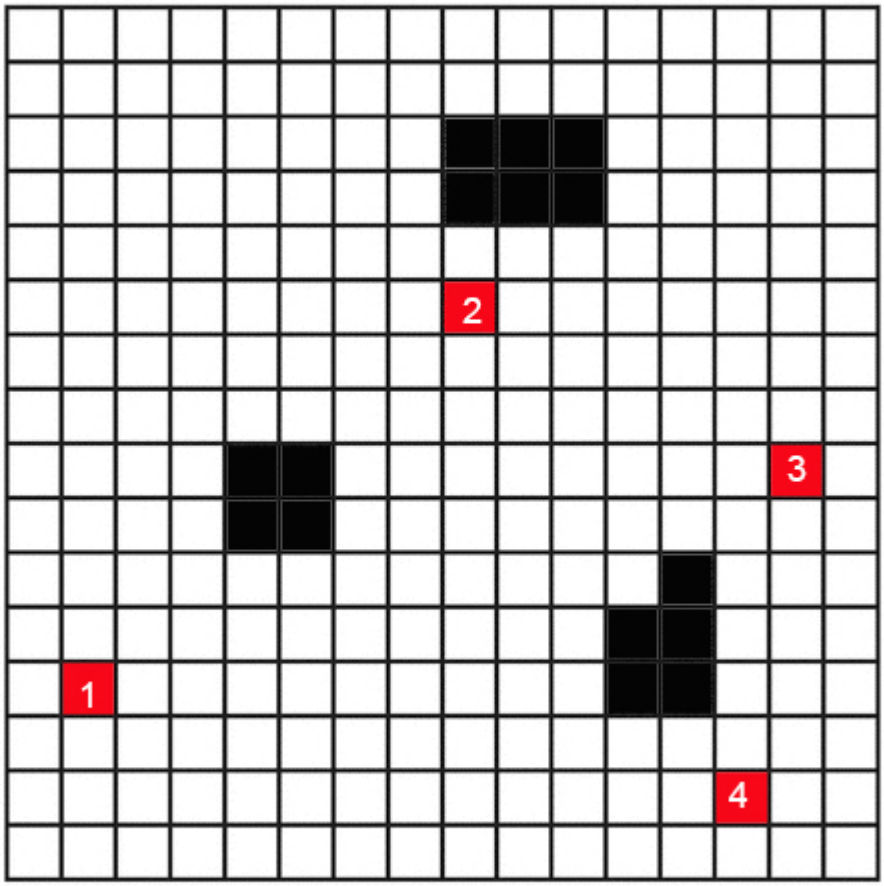


Figure 3.5 Example of waypoints in Grid-Based Navigation

to waypoint 3, and finally from waypoint 3 to waypoint 4. Black grid squares indicate areas that are not walkable and must be pathed around. The use of waypoints allows the agent's path to be shifted in the middle to increase believability, rather than relying upon the most direct path between two points.

The player character's location was converted to a series of waypoints on a grid of walkable terrain, with grid squares 1 unit in size. The character was assumed to be on the ground unless jumping, so the vertical axis information in the dataset was discarded. Jump height and distance were handled intrinsically by the physics portion of the game engine. The most common path taken by the player was then calculated separately from the players' actions. This path was used in conjunction with the Non-Hazardous SCMDP to determine the player's most likely direction of movement when the player's character was not being attacked. The agent's movement followed the most common path, utilizing the navigation routines included in the Unity API. This ensured that the agent followed the most likely path, while allowing some variance based upon the size of the character, the size of the grid square, and the calculation of the agent's proximity to its target square.

For the Hazardous SCMDP, the player agent's movement was limited to an adjacent grid square, with the direction of movement based upon transition probabilities determined using the MAGIC algorithm. As with the Non-Hazardous SCMDP, movement could occur at the same time as another action, such as Crouching or Attacking. Once the player agent was no longer being attacked, the closest waypont on the most common path through the area was found, and the agent returned to that path.

3.5 Test Results

As we had no predefined model to use as a basis for comparison, we had to develop another method of determining the outcome of our process. Although the validity of the Turing test has been recently called into question [25], there was no clearer way for us to determine the believability of a model built using real-world data. Therefore, to determine the effectiveness of our technique, we used a variation of a Turing test. We randomly chose one human player's actions, and used the Fraps program to record the playback as a video file. We then recorded our bot's actions in the same areas. We

uploaded both files to Youtube, and created an online survey that asked participants if the bot was the first video, the second video, or if they were unable to determine which was the bot and which was the human player.

Sixty-three people participated in the test. Participants were comprised of seniors in Computer Science at the University of South Carolina, freshmen in Engineering at the University of Wisconsin, and people who were recruited through Facebook. Of the sixty-three people that participated, thirty-three either chose the wrong video or indicated that they could not determine which one was the bot, and thirty chose the correct video. As more people were either incorrect or unable to determine which video was the human player, our test appeared to indicate that the technique we used produced convincingly human behavior. There were several problems, however, with the validity of our test results. Participants were able to communicate about the test before responding, allowing them to influence each others' answers. The problem was exacerbated by positional bias, as participants are more inclined to give the first answer when presented with a two choice question, and the addition of the "unable to determine" answer gave the participants an easy way to avoid deciding between the two, inclining some of those that may have chosen one or the other to avoid making a decision [52]. There was also no way to determine if the participants had actually watched the videos before answering the question, or had communicated with someone that had already answered the question.

The use of third-person view instead of first-person also was a potential issue. Actions that looked normal from a first-person perspective could appear strange when viewed in third-person, which made the character appear less believable even if it was the playback of a human player's actions. This was particularly the case when a player got stuck on an object briefly. From a first-person perspective, it was barely noticeable, but in a third-person view, it appeared as if the animation was stuttering. Animation problems could have been avoided by using first-person perspective during

the test, but we decided to use third-person so that the participants could see the agent's actions more clearly. While this made the actions of the agent more obvious, it may have contributed to the believability of the agent.

It was interesting to note the difference between the responses of the Computer Science students and the other testers. The students, who had more in common with the people that originally played the game that the player agent was based on, were more likely to be more uncertain as to which video was the bot and which was the human. Some participants from the non-student group that chose correctly commented upon the path that the bot took through the simulation, mentioning things like, "I would be more likely to explore the room more." This indicated both the importance of similarity to the target audience, and the importance of the player agent's path through the simulation. The most common path taken by the students playing the game was the shortest path to achieve the next goal. There was some indication, however, that the other testers were more likely to have additional goals for themselves, like exploring the rest of the game environment, rather than just accomplishing the goals given to them.

The bot that was created using only one SCMDP appeared to have erratic behavior. The lack of limitations on the bot's actions made it act in ways that were inappropriate to the situation, such as attacking at random or attempting to interact with objects that were not present. This made the bot appear much less believably human, as was apparent in the reaction of others when the bot's behavior was displayed to them. Updated examples of the bot without rules can be seen at `https://youtu.be/KqOPKT_lC1g`, `https://youtu.be/76LpErvsDRo`, `https://youtu.be/m71TRLVSWQQ`, and `https://youtu.be/qNbCJ9e3GlM`. An updated example of the bot with added rules can be seen at `https://youtu.be/TkQr4jQ5M9s`.

The use of four SCMDP's was not helpful, and in some cases not possible, because the rules it produced were too weak to make any difference in the end result. Players

were not injured enough to change their behavior based upon injury status, and players that picked up ammunition did so whether or not they already had it. Having too little data to create reliable SCMDP's using these rules showed that additional information is not always helpful. Instead, it can result in extra calculations that do not improve the end process. In our simulation, accounting for some *internal knowledge* resulted in *weak rules* that did not improve behavior models, or situations that did not allow creation of SCMDP's at all.

While our test results were not entirely conclusive due to problems with the way in which the test was conducted, it did appear that the best results were obtained by the agent that adapted to change in the simulation environment. The bot that used two SCMDP's based upon environmental change allowed the limitation of the agent's actions to those that were appropriate in the given environment. This made the agent's actions more believable and appear less random than the use of a single SCMDP. Incorporating the average time taken by the player to complete each action also made the agent appear more believably human, as agents can act much more quickly than human players. Combining timed actions and simple rules resulted in an agent that appeared to be more believably human. However, this agent was more believable to testers that were similar to those that originally played the game than testers from other backgrounds.

3.6 Conclusions

Our calculation of the transition values using both two and four SCMDP's showed that the players tended to act in a consistent fashion, regardless of the change in their character's health or the amount of ammunition they had. Using this added information only incorporated an additional clause in the decision-making process and increased the number of SCMDP's stored in memory. This appears to underscore the fact that we cannot make assumptions about human decision processes. The

things that appeared, at first, to be important considerations when the player had to determine his next action were not the most important factors. Change in the simulation enivronment by the introduction of a hazard in the form of an enemy agent affected player behavior, while the player character's health and remaining ammunition did not. Simple adaptation to a change in the simulation environment, however, made the agent's actions more believably human. It limited actions to ones that were more appropriate in the current context, rather than making them appear to be randomly chosen actions. The rules added were based both upon the results of the experiment and human knowledge of the simulation. The process was partially automated, but it still required interpretation by a domain expert.

Positional information was crucial in the creation of a believably human agent. Some actions are only appropriate in certain locations, making an agent that used these actions in inappropriate places appear unbelievable. Believability was also greatly affected by the agent's path. Even agents that take appropriate actions did not appear believably human when their movement was too erratic. We were able to find the agent's most common path programmatically and assign appropriate waypoints by using an HMM. However, as there were a minimal number of actions that were location-based, actions were linked to locations based upon human knowledge of the simulation. As these actions occurred in a consistent location, it should be possible in the future to also link the actions to location programmatically if necessary.

The most obvious limitation of this process, apart from the knowledge of a domain expert, is the increase in complexity based upon the number of conditions. With only 2 conditions, the increase is minimal. However, if too many conditions are taken into account, the additional time taken to switch between SCMDP's and the amount of memory required to store all of the SCMDP's simultaniously could outweigh the benefits of increased believability.

The target audience for the simulation is another important factor. Testers once again responded more favorably when the actions taken by the bot were more like the actions they would take. Testers that were dissimilar to the original game players that provided the information to create the bot were less likely to find the bot's actions realistically human.

Chapter 4

Generation of Multiple Models From a Single Data Set

4.1 Introduction and Related Work

In Chapter 2 and Chapter 3, we discussed creation of a single model from raw observational data. As we have seen in the previous chapters, this can work well if the behaviors being modeled have enough similarity. However, there are times when it is more appropriate to have multiple models, as we have seen with both the video game players in Chapter 3 and the nurses in Chapter 2. If there is more than one distinct behavior pattern, a model based upon the average observed behavior is not an accurate representation of human behavior in the given simulation. In the case of multiple distinct behavior patterns, it is more accurate to create multiple models.

Graph similarity measurement can be a difficult problem. Depending upon the size and type of graph, and the method of measurement, it can frequently be an NP-hard problem. However, if enough is known about the graphs, it becomes a much simpler problem [63]. The graphs created using the MAGIC algorithm have a given set of possible nodes and weighted edges, making them much more simple to compare than unweighted graphs with an unknown set of nodes. Graphs that are similar can be clustered, allowing us to use the MAGIC algorithm with each cluster to create multple models. These models can then be imported into a simulation in order to test their believability.

4.2 Problem Description

The MAGIC algorithm, shown in Chapter 2, uses raw data to automatically generate a single decision process which represents the average behavior of the entire population exhibited in the data set. Unfortunately, the average of an entire population is not a good representation of a single agent when the humans in the data set do not display behavior patterns that are sufficiently similar. For instance, in the nursing simultation, some nurses administer medication and perform other care during a single visit to the patient, while others separate these activities. If we take the average of these behaviors, we end up with a nurse agent that does not conform to either behavior pattern. Due to the difference in behavior in a single data set, we needed to find a way to generate mutiple models, each representing a different type of agent. Another example is the video game data obtained in Chapter 3, in which we wanted to obtain one model for the speed running players, and one for the attacking players.

Our solution to this problem is to cluster the data based on similarity between observations, which we determine by using the MAGIC CLASS algorithm, which we will describe in Section 3. We then use the MAGIC algorithm to create a separate SCMDP for each cluster, therefore obtaining multiple types of agents from a single data set. These behaviors can then be used in a simulation in order to determine their believability.

4.3 Research Approach

The decision process generated by the MAGIC algorithm is an SCMDP, or Sequential Compressed Markov Decision Process. This decision process can be viewed as a weighted graph. The nodes of the graph are the actions that can be taken by the agent, and the edges between them are weighted based upon the transition probability of one action following another. For our purposes, graphs can be considered similar

if they contain the same nodes, and the weights on the edges between these nodes are sufficiently similar.

In our particular problem, similarity measurement is simplified. The number of nodes in the graph is limited to the set of actions that can be taken by the agent. The graph of the average behavior contains all of the actions, and the graph of each individual's behavior contains a subset of those actions. As the edges between nodes are weighted, it is possible to use the edge weights as a distance metric, as follows:

Given two weighted, directed graphs, $G = (V, E)$ and $G' = (V', E')$ where $V' \subset V$, and given weighted edges $w(u, v) \,\forall\, V \in G$, and $w'(u', v') \,\forall\, V' \in G'$, the distance vector can be found by using the formula

$$\vec{d_v}(G, G') = \sum_{(u,v)\in V} w(u, v) - w'(u, v) \tag{4.1}$$

We let $w(u, v) = 0$ if there is no (u, v) in E'. For each edge (u, v), v can be referred to as the **destination vertex**. Calculating the distance for each destination vertex in G, we obtain a distance vector $\mathbf{d}$ with k features, where k denotes the number of destination vertices in G.

While there are other forms of similarity measurement that can be used for small graphs, such as Graph Edit Distance [14], these forms of measurement use either undirected graphs or graphs without edge weights. The use of edge weights makes a distance vector a much simpler method of determining distance, as the distance between two graphs can be easily computed by using the differences between edge weights, making the distance between two graphs computable in $\mathcal{O}(|V|^2)$ time, where $|V|$ is the total number of nodes in the graph. This means that it is possible to find the distance between the graph of an individual observation and the average graph created by the MAGIC algorithm in $\mathcal{O}(m|V|^2)$ time, where m is the number of observations.

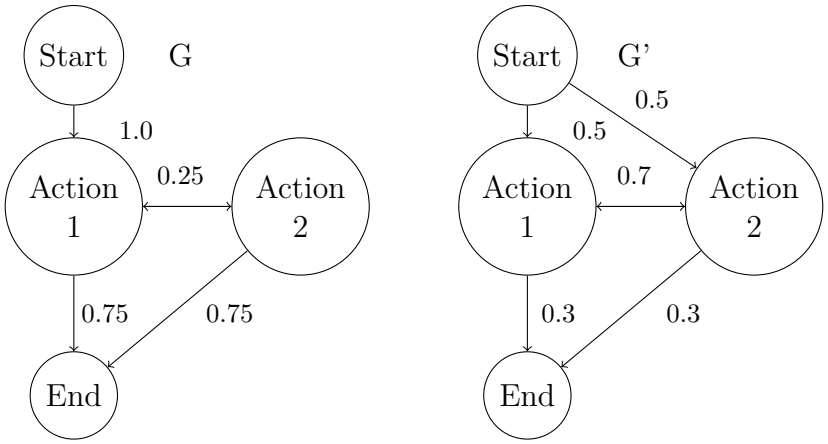


Figure 4.1 Example SCMDP's for use with MAGIC CLASS algorithm

Table 4.1 Example of graph similarity calculations for use with MAGIC CLASS algorithm. Rows and columns are the $v \in V$, and each cell contains $w(u, v) - w'(u, v)$ for each pair.

To node	Start	Action 1	Action 2	Total
Action 1	1 - 0.5 = 0.5	0 - 0 = 0	0.25 - 0.7 = -0.45	0.05
Action 2	0 - 0.5 = -0.5	0.25 - 0.7 = -0.45	0 - 0 = 0	-0.95
End	0 - 0 = 0	0.75 - 0.3 = 0.45	0.75 - 0.3 = 0.45	0.95

To find the similarity measurement between each observation and the graph obtained by using MAGIC with all observations, which we will refer to as the **average graph**, we first need to use the MAGIC algorithm to create a single SCMDP that represents the average behavior obtained from all of the observations. We then need to use the MAGIC algorithm on each individual observation, so that each observation has its own SCMDP. We can find the distance between the average graph and each observation's individual SCMDP by creating a table of distances between weighted edges for each node, and then adding the values in each column to find the total distance for that particular node, as shown in Table 4.1, which shows the calculated distance vector for the example graphs shown in Figure 4.1.

To calculate the distance between nodes, we subtract the edge weight of the edges going in to a node in an individual graph from the weight for the equivalent edge on the average graph. The resulting distance is placed in a table, as shown in Table 4.1. Once the total distance for each column is calculated, the graphs can be clustered using this vector of distances. The resulting clusters can then be used to create SCMDP's by using the MAGIC algorithm with each cluster used as the input data, creating one SCMDP for each cluster. For our purposes, we use both the K-means algorithm [46] and the Mean Shift algorithm [15], as both K-means and Mean Shift are considered good clustering methods when using distance as a feature. K-means is frequently used when there is some idea as to how many clusters should result, and Mean Shift is frequently used when there is no advance knowledge as to how many clusters are present. In our test, we compare the two methods using the synthetic data to determine both the effectiveness of MAGIC CLASS, and the efficency of the two classification algorithms when used with MAGIC CLASS.

4.3.1 K-means

The K-means algorithm, also known as Lloyd's algorithm, attempts to assign data points to clusters by minimizing the distance of those points to the centroid of the cluster, using the Euclidean distance formula

$$d(\mathbf{x}, \mu_i) = \|\mathbf{x} - \mu_i\|_2 \tag{4.2}$$

where $\mathbf{x}$ is the location vector of the current data point, and μ_i is the centroid of the assigned cluster. The algorithm initializes a centroid for each of the k clusters, then repeatedly assigns each data point to the nearest cluster and recalculates the cluster's centroid, until all data points are assigned.

After all data points are assigned, the distance between each point and each cluster mean is calculated, and data points are moved if necessary. Cluster centroids

```
MAGIC-CLASS(G, G', S)
 1  T = ∅ // List of transition probabilties
 2  for s ∈ S
 3       for s' ∈ S // For each possible transition
 4            T' = 0 // Initialize to 0
 5            for s' ∈ S
 6                 if ¬∃_{s'∈G}
 7                      s'_i = 0
 8                 if ¬∃_{s'∈G'}
 9                      s'_j = 0
10                 T'_s = s'_i − s'_j // Find the difference between the edge weights
                   going to the node
11                 T' = T' + T'_s
12            T.append(T')
13  return T
```

Figure 4.2 The MAGIC CLASS helper algorithm. G is the weighted directed average graph produced by the MAGIC algorithm, $G'[]$ is the weighted directed graphs for an individual observation produced by the MAGIC algorithm, and S is the list of possible actions s. This results in a vector used to cluster the individual graphs.

are then recalculated. This is repeated until no data points are reassigned. The K-means algorithm is an iterative algorithm, so its time complexity is $\mathcal{O}(kmn)$, where n is the number of data points, k is the number of features, and m is the number of iterations. A pictorial example is shown in Figure 4.3.

4.3.2 Mean Shift

The Mean Shift algorithm [26] is based on the principle of Kernel Density Estimation. Using an initial estimate and the chosen kernel, which is frequently the Gaussian kernel, the algorithm determines the weighted distance to the mean for each point. The points are then shifted towards the point of highest density, and the centroid is recalculated. This process is repeated until they converge.

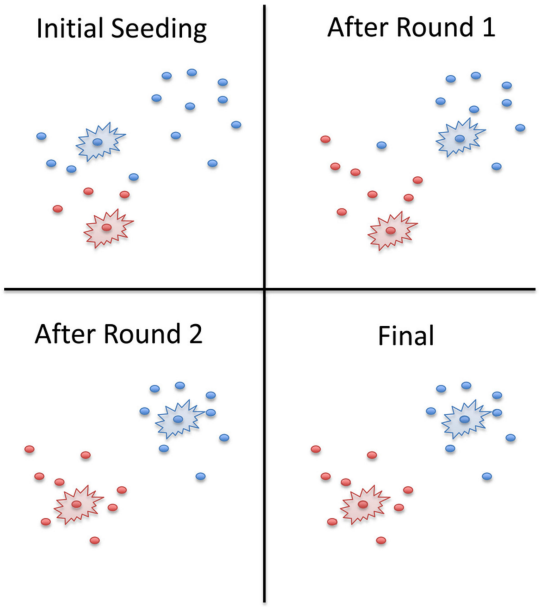


Figure 4.3 Example of K-means clustering
Image taken from [67] and used under Creative Commons License 4.0.

Given a set of data points x_i , a neighborhood $N(x)$ of points within a given distance, and a kernel $K(x_i - x)$, the weighted mean of the density in the kernel is calculated using the formula

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)}} \tag{4.3}$$

As the Gaussian kernel can be represented by $\phi(x) = e^{-\frac{x^2}{2\sigma^2}}$, the weighted mean density when using the Gaussian kernel is $K(x_i - x) = e^{-c||x_i - x||^2}$

Since the mean shift $x \leftarrow m(x)$ is calcuated simultaneously, a fixed kernel width, or bandwidth h is used to smooth the data.

Unlike K-means, Mean Shift does not require knowledge of the number of clusters. Instead, it relies on the kernel that is chosen, and the bandwidth, or window size. The choice of bandwidth is important, as it has a great impact on the number of clusters. A very small bandwidth can result in a large number of small clusters, while a large bandwidth can result in a small number of large clusters.

Figure 4.4 Example of Mean Shift with $h = 0.8$
Image taken from spin.atomic.com

The effect of bandwidth choice on the Mean Shift algorithm can be viewed as a topographical map, as seen in Figures 4.4 and 4.5. In the first instance, the bandwidth is small, resulting in many small, spiky clusters. In the second instance, the bandwidth is large, resulting in fewer large, smooth clusters.

The classic Mean Shift algorithm has a time complexity of $O(mn^2)$, where n is the number of data points and m is the number of iterations. This makes Mean Shift more time-intensive to use than K-means if there are a large number of observations.

4.3.3 Comparison Between Clustering Methods

Mean Shift can be effective when the number of clusters is unknown, as can be the case in real-world data. It is, however, generally slower than K-means ,and may not work well if there are too many outliers or too many local maxima [15]. Therefore, K-means may be more advantageous to use for classification of larger graphs, as long as the approximate number of behavior patterns can be determined. In the case of both algorithms, they may need to be run several times in order to determine the

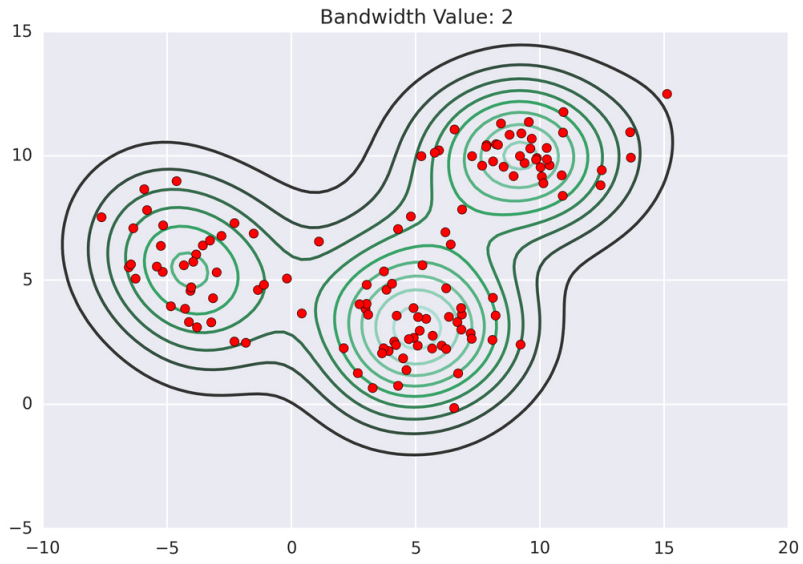


Figure 4.5 Example of Mean Shift with $h = 2$
Image taken from spin.atomic.com

best K, in the case of K-means, or the most appropriate bandwidth, in the case of Mean Shift.

The need to create and compare models from the results in order to determine effectiveness makes the use of K-means far more attractive if there is any indication as to the number of models that should result. This also means that if there is not a significant enough similarity between observed behaviors so that they can be grouped, it is better to obtain more data, if possible, than to attempt to classify the observations. Given a small enough bandwidth, data that is too dissimilar will result in a large number of models.

There is also the problem of data similarity. If there are too many **borderline** models, or models with graphs that are very close to the average graph, these models will become more difficult to classify. If Mean Shift is used as the final classification algorithm, it is possible that the models closest to the average graph will be classified as an additional cluster. This result may be good if the number of models is unknown

and similarity is the only requirement, but will be problematic if the average graph is meant to be the division point between models.

4.4 Testing

As we do not have models for a comparison of accuracy when attempting to properly classify real-world data, we needed to create our own synthetic model. We used this model to verify the correctness of the MAGIC CLASS algorithm, and to compare the use of the distance vector as a feature set with both clustering methods.

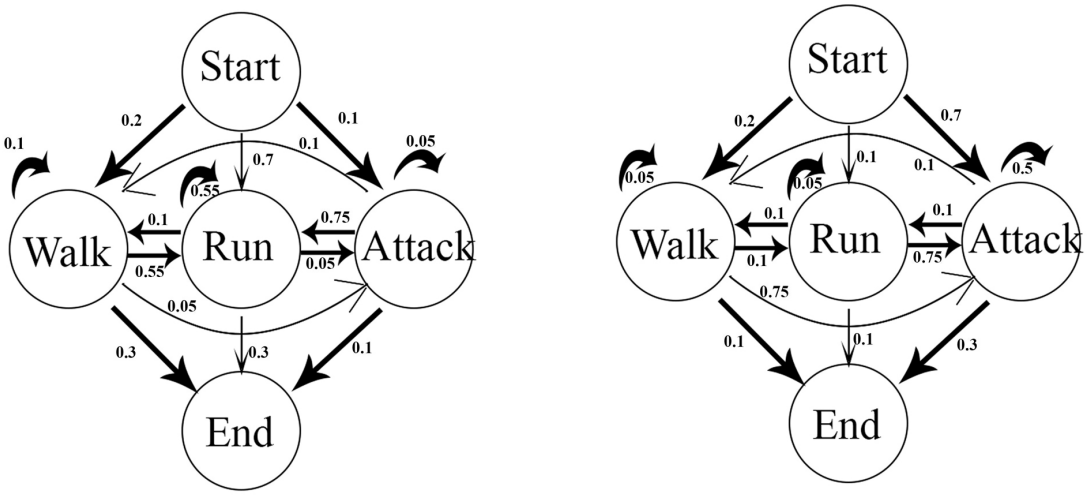
After verifying the accuracy of our process with synthetic data, we used the MAGIC CLASS algorithm with the data obtained from the NoMadMan game, described in Chapter 3, to cluster the player data into two groups. We then used those clusters two create two different player agents from the same original data set.

4.4.1 Validation Using Synthetic Data

In order to validate our methodology, we created two distinct models of behavior, as shown in Figure 4.6. We used these models to create 5,000 random strings of observations from each of them in a random order, as shown in Table 4.2.

The number 1 or 2 appended to "start", shown in the sample data in Table 4.2, indicates which model was used to create the random string. The observations that we created were saved in a text file in comma delimited format. When using the file, we removed the "1" or "2" and stored them in a list to indicate the appropriate cluster for that observation. We then used the MAGIC algorithm to find the average graph for the behavior models, shown in Figure 4.7, and the graph for each individual observation, an example of which is shown in Figure 4.8.

We calculated the distance between the graph for each observation and the average graph using the MAGIC Class algorithm to create a table of distances to each node, and added those distances to create a distance vector, shown in Table 4.3. We stored



Decision Process 1 **Decision Process 2**

Figure 4.6 Graphs used in synthetic validation of MAGIC CLASS

Table 4.2 Example of synthetic data used to verify the accuracy of MAGIC CLASS

start1, walk, end
start1, walk, end
start2 ,attack, attack, end
start1, walk, run, run, run, run, walk, end
start1, run, end
start2, walk, end, run, attack, attack, end
start2 ,attack, attack, attack, walk, end
start1, run, end
start1, run, walk, end
start1, run, end

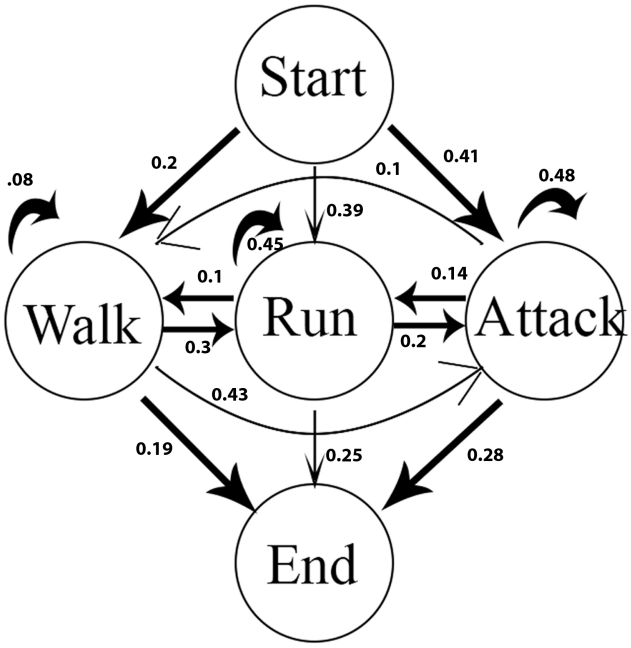


Figure 4.7 Example average graph

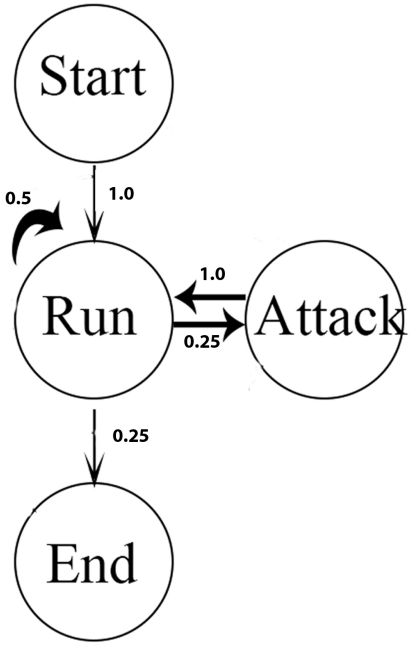


Figure 4.8 Example graph of individual observation

Table 4.3 Example of distance calculations

	start	walk	run	attack	total
walk	0.2	0.08	0.1	0.1	0.48
run	-0.61	0.3	-0.05	-0.86	-1.22
attack	0.41	0.43	-0.05	0.48	1.27
end	0.0	0.19	0.0	0.28	0.57

the resulting distance vectors for each observation and used the K-means algorithm to classify the results, setting k to 2. Using the same distance vectors, we used the Mean Shift algorithm with a Gaussian kernel and an estimated bandwidth derived from the first 500 data points to classify the data.

We ran the program 10 times, using 10 different sets of random data created using the same models. In each run, we generated 5,000 observations for each model in random order, and classified them using the MAGIC CLASS algorithm. We used the distance vectors created by the algorithm to cluster the resulting distance vectors, using both the K-means and the Mean Shift algorithm for comparison. We measured the accuracy based upon the label provided by the clustering algorithm, using the formula

$$Accuracy = \frac{(\sum c \in C)}{n} \tag{4.4}$$

where C is a cluster, c is a data point correctly assigned to that cluster, and n is the total number of data points. The results, shown in Table 4.4, varied depending upon the random data. The K-means algorithm had a very consistant success rate, showing us that if the number of behavior types is known, K-means is the most consistently accurate. The Mean Shift algorithm, on the other hand, was either extremely accurate, or created too many clusters.

Due to the reliance on bandwidth size for Mean Shift, it makes sense that if there are more borderline clusters, or clusters of graphs that are close to the average graph,

Table 4.4 Accuracy using K-means algorithm with k=2 and Mean Shift algorithm over 10 runs.

Run Number	K-means Accuracy	Mean Shift Accuracy	Mean Shift Clusters
1	94.73%	88.59%	3
2	95.12%	99.05%	2
3	95.55%	77.65%	3
4	95.06%	99.99%	2
5	94.43%	99.77%	2
6	95.93%	78.18%	3
7	95.10%	99.50%	2
8	95.04%	77.96%	3
9	95.04%	77.47%	3
10	95.64%	99.06%	2

they will be grouped together. In cases where the correct number of clusters were found, however, classification was extremely accurate. As we have no prior knowledge of the number of clusters that should result when using Mean Shift for classification, we have to use the MAGIC algorithm to create models from the clusters, then use the simulation to visualize the results in order to determine if the models are believable. Then, if the results are not believable, we must adjust the bandwidth and repeat the process. While this is possible, it is not ideal, in that it can increase the amount of time it takes to find believable models.

As we are attempting to classify the video game data into two groups, we decided to use the K-means algorithm. The accuracy of the results of our testing using synthetic data was consistent when using the K-means algorithm, as it relies upon the choice of k, rather than an estimated bandwith. We have seen that, while the Mean Shift algorithm can provide more accurate results, it can also provide results that are far less accurate if the estimated bandwidth is poorly chosen.

4.4.2 Multiple Models Using Real-World Data

After testing MAGIC CLASS with synthetic data, we wanted to see how it would function with real-world data. We decided to create multiple player models from the video game data that we obtained during our prior testing in Chapter 3. In many first-person shooter games, there are players that prefer to "speed run", or attempt to reach the final objective as quickly as possible. Speed runners will often compete with each other in order to achieve the fastest completion time. Players that speed run tend to skip as much unnecessary content as possible, including not fighting enemy characters if they can avoid it. Instead, they will run past the enemies, attempting to avoid combat scenarios that will slow their progress. Players that are not attempting to speed run, however, are more likely to fight the enemies, rather than simply to run past them. Since speed runners are common enough in video games, we hypothesized that we could use MAGIC CLASS to separate them from the group of players that approached the game normally.

After using the MAGIC CLASS algorithm, we used the K-means algorithm with k=2 to separate the observations into two clusters. We used the MAGIC algorithm to create two models, one from each cluster. The resulting models, however, were not divided into speed runners and normal players. Instead, they were divided into the people that saw the AI guard and started combat before it "saw" them, allowing them to attack the guard without being attacked in return, and the people that ran into the room and then began combat with the guard. The players that began combat after entering the room were much more likely to attack repeatedly, while the ones that attacked the guard from a distance had no need to do so, since they were able to kill the guard before it responded. This once again showed that our initial assumptions about the players behaviors were not necessarily correct, and that classifying the models programmatically provided a better result than attempting to do so based upon our initial hypothesis.

As we have already tested the believability of models produced by the MAGIC algorithm in Chapter 3, we did not conduct the same test again. Instead, we created a demo showing the difference in the behavior of the two models and recorded the results in video format. We used what we learned in Chapter 3 to make the end results more believable by finding the average minimum distance from the guard when attacking, so that the player agent would not continue to advance towards the guard while attacking if the guard was too close. In order to differentiate the videos from the ones used in the Chapter 3 test, the background was replaced with the background used in the final scene of the game. Locations of objects and enemies were replicated so that the bot's path would remain accurate. The resulting videos of the bot created from Cluster 1 can be seen at `https://youtu.be/0IkrIW-7dJ0` and the bot created from Cluster 2 can be seen at `https://youtu.be/P06Oc_8BEis`.

4.5 Conclusions

We have shown that, while this method can be used to develop more than one decision process from the same set of raw observational data, the resulting processes must be distinctly different from one another in order to obtain believable models. Processes that are too similar will be classified as the same behavior. This is particularly noticeable in the difference between the results when using Mean Shift rather than K-means as a classification algorithm.

If the number of behavior patterns is unknown, we have seen that it is likely that multiple attempts will have to be made in order to classify the behaviors appropriately. Depending on the size of the graph and the number of observations, however, we have found that this may still be more efficient than attempting to classify behavior patterns by hand.

Since the K-means algorithm requires prior knowledge of the number of clusters, it is more likely to be useful in situations where it is known that there will be distinctly

different behavior patterns, such as in the nursing simulation. In the nursing simulation, some nurses administered medication and did other tasks, such as checking on injuries, during the same round. Other nurses separated their tasks, only administering medication during one round, then performing their other tasks during the next. Since the behavior patterns are distinctly different, and it is known that there are 2 patterns, the K-means algorithm should suffice to classify them. The advantage to this is that K-means tends to run in a shorter amount of time than Mean Shift.

On the other hand, if it is known that there is a significant difference in behavioral patterns, but not the number of patterns, the Mean Shift algorithm may be more appropriate to use. As it takes longer to run, it is more likely to be useful if the graphs being compared are small. In either case, it is probable that the results will have to be viewed in a simulation in order to determine the models' believability.

The effectiveness of this technique is greatly limited by the size of the graph and the number of observations. If the graph is large, and there are not enough observations, as was the case in the nursing simulation, it is likely that it will be difficult to obtain accurate results. If there are a sufficient number of observations and the graph size is small enough, however, we have determined that it is possible to use graph similarity measurements to obtain multiple decision processes from one data set.

Chapter 5

Possible Improvements and Future Work: Pathfinder Agents

5.1 Introduction and Related Work

Pathfinding has been a major area of study in artificial intelligence for a great many years. The focus, however, is frequently on finding the optimal path in the least amount of time. Instead, our focus is finding the most believably human path, which is not necessarily the path that is most optimal. Humans take paths that allow them to achieve specific goals or to perform certain actions. In order to create a believable player agent, we need to use observed location data and prior knowledge while finding an appropriate path in order to increase the agent's believability.

As we have seen in the previous chapters, believability is not only dependent upon the simulation itself. It is also dependent on the expectations of the observer. When groups of people with differing expectations are shown the same video, the group that expects the behavior depicted in the video is more likely to find it believable. In the case of a simulation like the game simulation, the agent's path is important in determining the agent's believability.

5.2 Research Problem

During the tests in Chapter 3, we learned that an agent's path can be a very important component in the agent's believability. Agents that only use minor variations in the same path eventually become less believable. As a demonstration of future work, we

would like to show some possible improvements to the agent's pathfinding ability that make the results more believable to the target audience.

As multiple calculations to determine a variable path at runtime can be costly in terms of time, we would like to make the calculations prior to the portion of the simulation that will be viewed by testers. In order to do this, we propose to simulate a walk through the simulation by using invisible primitives, taking into account the variance in the original path results, allowing for alterations to the agent's path while keeping within the bounds of the original observed location data. Using primitives to walk through the simulation also allows us to anticipate and avoid collisions by adjusting the agent's path in advance, making it unnecessary to check for collisions with stationary objects at runtime.

5.3 Research Methods

In Chapter 3, we used a Hidden Markov model to find the most common path taken by players during the data collection phase. We then created a player agent that used that most common path, deviating only slightly based upon speed of motion and current action. We used Unity's built-in Animation controller and the NavMesh algorithm along with the most common path to determine the player agent's movements. In this chapter, we suggest another way to determine movement that provides a greater variation in movement in order to make the agent appear more believable.

As an example of potential future work, we introduce **Pathfinder Agents** that calculate the player agent's path at the beginning of the simulation, prior to the player agent's movement. These agents use the locations in the most common path with the greatest variance to create alternate, possible paths, increasing the chance that the player agent's path will vary when the simulation is run multiple times. The Pathfinder Agents are not visible to the observer so that they do not make the simulation less believable.

5.3.1 Pathfinding

The introduction of simple rules to allow agents to use a subset of behaviors can increase believability, as shown in Chapter 3. However, certain actions are tied to specific locations in the simulation. If these actions are taken in other locations, it can significantly decrease both the believability of the agent, and the accuracy of the simulation.

Believability is also determined by the expectations of the observers, as seen in the tests in Chapters 2 and 3. In Chapter 2, the observers were clinicians, and the nurse agent's movement patterns followed a pattern that was designated by hospital guidelines. In Chapter 3, there were two different sets of observers. The observers that were computer science and engineering students, who were more similar to the group that originally played the game, found the player agent's actions more believable. The observers that were recruited via Facebook, who were not gamers, found them less believable. Some of these observers noted that, while they were still uncertain which video was the recording of the bot, they guessed based upon the path that the bot had taken.

The agent's movement pattern through a simulation is important, and is also a common problem faced in the video game industry [20]. Agents that do not move in a way consistent with human movement patterns appear to be bots, and are more easily recognizable [40]. We want to improve the believability and accuracy of our simulation using location data by further limiting actions to more specific locations, and by using pathfinding in conjunction with the decision process created using the MAGIC algorithm, shown in Chapter 2. In this chapter, we use an HMM to find the agent's most common path and the locations with the greatest variance from that path. We then use Pathfinder Agents to find a movement path for the agent, which we use in conjunction with the decision process created by the MAGIC algorithm to create a more believably human agent.

The A* algorithm [39] is commonly used in two dimensional games and simulations to find the shortest distance to a target. It uses a best-first search combined with a heuristic, such as Manhattan or "city block" distance, to calculate the shortest distance to a destination. Given a weighted graph G, a start node $s \in G$, a destination node $g \in G$, and a heuristic $h(n)$, for each iteration of the algorithm, it finds the path that minimizes

$$f(n) = g(n) + h(n) \tag{5.1}$$

where $f(n)$ is the shortest distance from the destination node to the current node. The A* algorithm is implemented using a priority queue, which indicates which way to extend the path based upon the current shortest distance. While the A* algorithm is guaranteed to find the optimal path to a target if given an appropriate heuristic, the resulting path, when followed by an agent, can appear erratic. The linearity and sharp turns in a path found by the original A* algorithm can make the agent's behavior appear less believably human.

Pathfinding in two dimensions was improved by the Theta* algorithm [61], which allows any node to be a parent to any other node. It adds a smoothing factor to the path in the form of an angle range with an upper angle bound and a lower angle bound for each node. In this manner, the agent makes fewer sharp turns, and its path appears less robotic and more believable [66]. The difference between paths found by the A* algorithm and the Theta* algorithm can be seen in Figure 5.1.

The A* algorithm and its any-angle variants [62], such as the Theta* algorithm, do not perform as well in a 3D environment [55]. The addition of a third dimension makes the calculation much more complex, particularly due to additional movement restrictions, such as the additional need for collision detection and the limitation of vertical-axis movement in gravity-simulated environments. The possibility of three

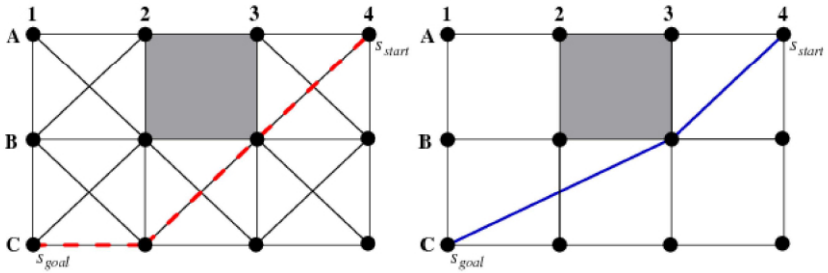


Figure 5.1 Path found by A* algorithm versus Path found by Theta* algorithm
Image taken from [66]

dimensional movement requires a different algorithm to find an appropriate path for an agent to reach its destination.

One of the most commonly used methods of pathfinding in 3D is the navigation mesh [81], or NavMesh. A NavMesh is created by marking polygons as accessible or inaccessible. Rules are set as to the distance an agent can climb and the steepest angle the agent can ascend or descend. Surfaces that can be navigated by the agent are referred to as **walkable**. The shortest path to the goal is calculated by using a combination of the A* algorithm or one of its variants, such as the Theta* algorithm, with the marked walkable area and the height and grade rules. By taking height and walkable areas into consideration, and the use of collision detection while the agent is moving, agent movement appears more believable. As our game was built in 3D, and NavMesh is implemented in the Unity API, in the test performed in Chapter 3 we used NavMesh to calculate the agent's path at runtime. An example of the NavMesh used is shown in Figure 5.2.

5.3.2 Pattern Matching Using Hidden Markov Models

Since we wanted to find the most common path taken by human players, rather than the most optimal path, we used a Hidden Markov Model with the location data recorded in the test performed in Chapter 3. A **Markov Chain** is a statistical model

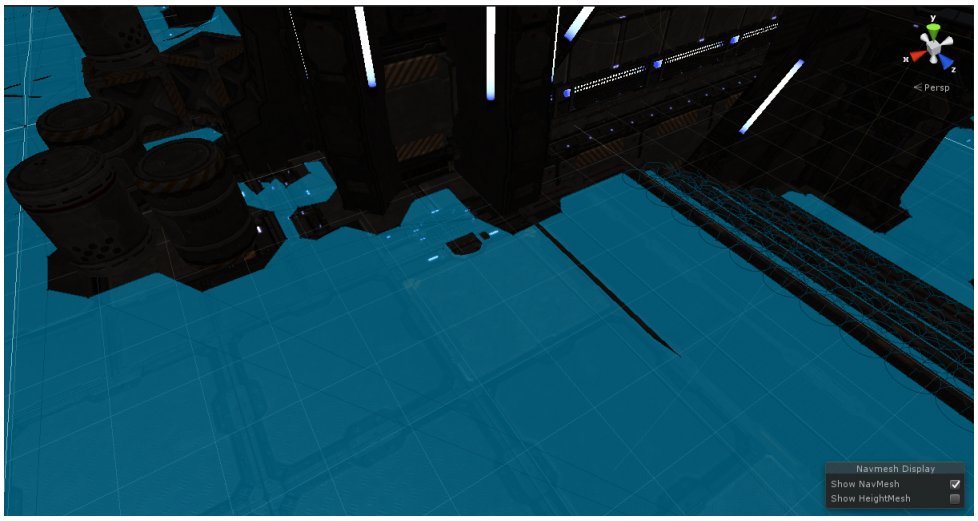


Figure 5.2 Example of a navigation mesh where the blue areas are walkable areas.

where the probability of a state is determined based only upon the previous state [3]. In other words, given a set of conditionally independant random variables $[X_i]$,

$$P(X_t = j \| X_0 = i_0, X_1 = i_1, ..., X_{t-1} = i_{t-1}) = P(X_t = j \| X_{t-1} = i_{t-1}) \qquad (5.2)$$

A **Hidden Marvov Model** is a Markov process where there is a set of observations, but the exact model is unknown. This is generally the case with real-world situations. **First Order Hidden Markov Models** make the same assumption as a Markov Chain, in that the probability of the transition to a particular state relies only upon the previous state. It is also assumed that the probability of an observation is independant. An HMM uses a matrix to represent the current state of the world depending upon the transition from the previous state. Given $X_t = 1...S$, where X represents a state, t represents the current time, and S is the number of possible states. Transitions can then be represented by an SxS matrix T, where the probability of a transition between states i and j can be calculated using the formula

$$T_{ij} = P(X_t = j \| X_{t-1} = i) \tag{5.3}$$

We then used the Viterbi algorithm [24] to find the most commonly used path, and used that path for our bot's movement. There was some variation in the path based upon the bot's speed and its relative position to it's previous target, but it's overall path was very similar each time the simulation was run. This could potentially make the bot's behavior less believable when viewed multiple times.

5.3.3 Pathfinding Research Approach

The method of data collection used in Chapter 3 allows us to record the players' exact locations when actions are performed. This is an improvement over the method used in the nursing simulation, which was described in Chapter 2. Due to HIPAA regulations, human observers were not able to enter rooms with the nurses, images could not be taken of nurses' activities, and nurses could not wear body cameras. This made the location data in the nursing simulation much more general, so that we were required to know which actions had to be taken in specific locations, and program the simulation accordingly. The use of hospital floor plans were of some help in increasing accuracy, but there was more room for error in the nurse agent's path.

The player agent can be given a more accurate choice of actions due to the recording of players' locations in each frame of the game. It is trivial to find actions that are only taken at specific locations when both the location and the action are recorded. This enables us to further restrict actions to specific locations, improving the believability of the player agent. For instance, we know that players can only pick up ammunition in locations where ammunition is present to be picked up. By a simple comparision of actions and locations, it is easy to determine the locations where ammunition is present, and restrict the "Get Ammo" action to those locations. As both the location and the action are recorded at the same time, we can determine

the actions and locations programatically, rather than requiring the knowledge of the actions that can only be performed at certain locations. If locations and actions could not be recorded simultaneously, we would require this knowledge in order to make the simulation appear believable.

Pathfinding is an important component when simulating human behavior, because movement is one of the most basic, required human behaviors to model. Inaccurate movement behavior makes the human model appear less believable. There have been many studies on simply finding the most accurate and believable path for human movement, such as "Pedestrians", a study in finding the appropriate movement behavior of humans in a crowd [4].

The use of the most common path and the NavMesh algorithm only allowed minor variations in the agent's path. This made the agent appear to walk along the same path every time the simulation was run, making it less believable when compared to multiple traces of human behavior. To increase the variability of the agent's path, we created Pathfinder Agents. Pathfinder agents are agents that run at the beginning of the simulation to create varying paths based on collected user data.

To calculate the path to be taken by the player agent, we use a Hidden Markov Model. The resulting location data, converted to waypoints, as shown in Chapter 3, is used by two Pathfinder Agents at the beginning of the simulation. The Pathfinder Agents determine the movement paths that will be taken by the player agent when it is using the non-Hazardous SCMDP, which was created using the MAGIC algorithm shown in Chapter 2.

Rather than using one of the more common methods of pathfinding for the player agent, we use the Pathfinder Agents to calculate the player agent's path at runtime. This creates a path for each run of the simulation that is specific to that run, but stays true to the observed data. In this way, we are able to vary the agent's path, while maintaining accuracy based upon our observations.

Pathfinder Agents are invisible primitives that move through a designated area in the simulation. Before the player agent begins moving, the Pathfinder Agents use the most common path and variance in that path, as determined by the HMM, to create and record a predesignated path for that particular execution of the simulation. In particular, the Pathfinder Agents focus upon the areas of the path with the highest degree of variance, and also include a minor chance for a slight drift from the most common path. Using invisible primitives allows us to account for adjustments due to potential collisions with objects, making traversal easier for the player agent. The Pathfinder Agents record their movement as lists of waypoints to be used by the player agent in each area. After the path for each area is created, the Pathfinder Agents are deactivated in order to conserve resources. By introducting a random amount of variance in the player agent's path that is based upon the variance between the paths taken by players, the player agent no longer takes an extremely similar path every time the simulation is run. This makes the player agent appear more believably human by introducing variety in the player agent's movement pattern.

While using the Hazardous SCMDP, the player agent's movement is also limited by proximity to the enemy agent. As we know that human players are not likely to attempt to run directly over an enemy that is attacking, and we know the location of the enemy agent, it is trivial to calculate the average distance players remain away from the enemy agent. This is incorporated into the player agent's movement, so that the agent will no longer continue to approach the enemy if it is too close. This also avoids collision between dynamic objects, as the player agent and the enemy agent will never be close enough together to collide, making collision detection between the two unnecessary while both agents are active.

5.3.4 Future Work : Pathfinding Demonstration

A new version of the simulation will be created, using the Pathfinder Agents and the additional rule about proximity to the enemy. We will use the background scene from Chapter 4 with the location information for objects and agents from the original simulation in Chapter 3. The bot's behavior will use the single model with 2 SCMDP's that was created in Chapter 3.

The Pathfinder Agents will travel through the simulation before the bot begins moving, dynamically creating a list of waypoints for the bot to follow. These waypoints will be determined using the points of greatest variance in movement in the original path, and were adjusted to avoid collision with stationary objects. Bot playbacks using both the old and new pathfinding methods will be recorded and uploaded to Youtube. Using Pathfinder Agents, we will be able to create models using multiple likely paths obtained from the original data.

5.4 Conclusions

Pathfinding is an important aspect when it comes to the believability of an agent's behvior in a simulation. Agents that make abrupt terms have been referred to as "robotic", and those that repond incorrectly to collisions or are erratic in their movements have been compared to humans that are drunk. If agents always take the same path in a simulation, and it is viewed repeatedly, the behavior of the agent can appear less believably human.

When using the most common path, as shown in Chapter 3, the results can be unbelievable due to the limited variation in the agent's behavior. The use of pathfinder agents is a reasonable attempt to correct this issue, as shown in our demonstration. It requires more time to calculate initially, but the initial calculation is not seen by testers. The results of that calculation are a more varied path that still uses the initial data, but provides different results when it is run multiple times, making it

potentially more believable to the target audience. However, if there isn't enough similarity in the paths taken in the data collection phase, the results could become erratic, making the agent's movements appear even more unbelievable.

If the results of the simulation are only viewed once, and the target audience is very similar to the people that were initially obvserved, adding Pathfinder Agents can be an unnecessary step, as it will simply increase the amount of time to create and run the simulation. Also, if it necessary for the agent to have a consistent path due to the constraints of the simulation, or if the path is required to be calculated during movement, the use of Pathfinder Agents would actually reduce believability. However, if the target audience expects some variance in the agent's path, and the results of the simulation will be viewed multiple times, Pathfinder Agents can increase the believability of the result by varying the movements of the agent while remaining within the path taken during the initial observations.

Conclusion

The simulation of human behavior is a complex problem. Although progress has been made in the study of human thought processes, we are still unable to fully explain how humans make decisions. This is further complicated by the fact that human decisions are influenced by different things at different times, ranging from social and cultural influences to the human's emotional state. These influences can vary from moment to moment, making modeling human decision-making a very difficult endeavor. To complicate matters even further, many of the psychological influences on human decision-making lack a common definition, and are therefore unable to be defined in mathematical terms. This presents a problem when we try to create a process that will imitate human behavior in a variety of simulations rather than in only one.

While we cannot observe human thought processes, we are able to observe human actions. Provided the humans that are being observed and the situations in which they are observed have a high enough degree of similarity, we can use those observations to find patterns that allow us to imitate human behavior programmatically, making it unnecessary to model human thought processes. This has become a more common method of imitating human behavior, and it is used both in simulations and games.

Agent-based modeling is an appropriate method of simulation, due to the complexity of human behavior. It provides a means to model interactions between human-based agents by modeling single agents, rather than the entire system. The emergent

behavior shown by the interaction of these agents can provide insight into the system as a whole.

Our method of progammically generating an agent's decision process solves the problems caused by communication difficulties between modelers and domain experts and reduces the inaccuracy we have encountered when creating human behavior models by hand. We have found that it is possible to create models more quickly and more accurately by creating them programmatically. We have also made it simpler to adjust models as desired, so that domain experts can view the results of their adjustments in a simulation environment without the need for extensive additional calculations.

We have provided an extension of the Markov Decision Process to be used in human behavior modeling. It is more useful than a traditional MDP because it allows for an action to be repeated a certain number of times, or until a set condition is met. By doing so, we have provided a way to model decision points, or places where the agent must obtain information from another agent or the environment in order to proceed. This, in turn, makes the human model more believable.

We have also determined that it is possible to have too much information, despite the fact that recording enough accurate information is very important to our process. Making assumptions about the reasons for particular types of behavior can be harmful to the creation of the model. In both the nursing and the game simulations, we found that there was a difference between our assumptions about behavior and the end model. In the nursing simulation, we found decision points that we had not previously considered. In the video game simulation, we found that players disregarded information that we assumed would be relevant. When dividing the player observations into two groups, we found that the difference in behavior patterns was different than the one that we had anticipated.

We have shown that our process works in differing types of simulations, provided there are enough observations and that there is enough similarity between the humans being simulated. We have also found a simple method to use our process in simulations with changing environments. Using simple rules, we were able to create more believable models.

In a given set of observations, there is sometimes more than one distinct behavior pattern, as we have seen both in the simulation of nurses administering medication in a hospital and in the behavior of players in a video game. By using a graphical version of the behavior process and a distance metric, we have shown that it is possible to classify the behaviors in order to create multiple models from the same set of observations. As these models are created from different, distinct behavior patterns, they are more accurate than an average of the combined behaviors.

We have also seen that the believability of the agent is greatly dependant upon the target audience. If the audience differs greatly from the humans being modeled, they are more likely to find the agent's behavior unbelievable, as the actions of the agent do not necessarily correspond to the actions that would be taken by the viewer. Simulations, therefore, must be tailored to the intended audience. This is not a problem for research-based simulations, as the viewer is likely to be a domain expert. It presents more of a problem in the case of video games or simulations that would be viewed by a wider variety of people. In these cases, additional steps, such as allowance for some variance in behavior, can make the agent appear more believably human.

Bibliography

[1] Rogelio Adobbati, Andrew N Marshall, Andrew Scholer, Sheila Tejada, Gal A Kaminka, Steven Schaffer, and Chris Sollitto. Gamebots: A 3d virtual world test-bed for multi-agent research. In *Proceedings of the second international workshop on Infrastructure for Agents, MAS, and Scalable MAS*, volume 5, page 6. Montreal, Canada, 2001.

[2] Philip Anderson. Perspective: Complexity theory and organization science. *Organization science*, 10(3):216–232, 1999.

[3] Theodore W Anderson and Leo A Goodman. Statistical inference about markov chains. *The Annals of Mathematical Statistics*, pages 89–110, 1957.

[4] Koji Ashida, Seung-Joo Lee, Jan M Allbeck, Harold Sun, Norman I Badler, and Dimitris Metaxas. Pedestrians: Creating agent behaviors through statistical analysis of observation data. In *Computer Animation, 2001. The Fourteenth Conference on Computer Animation. Proceedings*, pages 84–92. IEEE, 2001.

[5] Scott Barton. Chaos, self-organization, and psychology. *American Psychologist*, 49(1):5, 1994.

[6] Antoine Bechara, Hanna Damasio, and Antonio R. Damasio. Emotion, decision making and the orbitofrontal cortex. *Cerebral Cortex*, 10(3):295–307, 2000.

[7] William H. Bell, David G. Cameron, Ruben Carvajal-Schiaffino, A. Paul Miller, Kurt Stockinger, and Floriano Zini. Evaluation of an economy-based file replication strategy for a data grid. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 661–668, 2003.

[8] Darrin C Bentivegna, Christopher G Atkeson, and Gordon Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2-3):163–169, 2004.

[9] Thomas Berger. Agent-based spatial models applied to agriculture: A simulation tool for technology diffusion, resource use changes and policy analysis. *Agricultural economics*, 25(2-3):245–260, 2001.

[10] James R Bettman and C Whan Park. Effects of prior knowledge and experience and phase of the choice process on consumer decision processes: A protocol analysis. *Journal of consumer research*, 7(3):234–248, 1980.

[11] Keith Beven and Jim Freer. Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the glue methodology. *Journal of hydrology*, 249(1):11–29, 2001.

[12] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(3):7280–7287, 2002.

[13] Lothar Breuer. Introduction to stochastic processes. *Lecture Notes, Univ. Kent*, 2014.

[14] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3):255–259, 1998.

[15] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[16] Philip R Cohen and Hector J Levesque. Intention is choice with commitment. *Artificial intelligence*, 42(2):213–261, 1990.

[17] Kenneth Mark Colby. Modeling a paranoid mind. *Behavioral and Brain Sciences*, 4(04):515–534, 1981.

[18] Kenneth Mark Colby, Sylvia Weber, and Franklin Dennis Hilf. Artificial paranoia. *Artificial Intelligence*, 2(1):1–25, 1971.

[19] Diane J Cook, Michael Youngblood, Edwin O Heierman III, Karthik Gopalratnam, Sira Rao, Andrey Litvin, and Farhan Khawaja. Mavhome: An agent-based smart home. In *2013 IEEE international conference on pervasive computing and communications (PerCom)*, pages 521–521. IEEE Computer Society, 2003.

[20] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[21] Tom De Wolf and Tom Holvoet. Towards autonomic computing: Agent-based modelling, dynamical systems analysis, and decentralised control. In *Industrial*

Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on, pages 470–479. IEEE, 2003.

[22] Peter Deadman, Derek Robinson, Emilio Moran, and Eduardo Brondizio. Colonist household decisionmaking and land-use change in the amazon rainforest: An agent-based simulation. *Environment and Planning B*, 31:693–710, 2004.

[23] Michael W. Floyd, Babak Esfandiari, and Kevin Lam. A case-based reasoning approach to imitating robocup players. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*, pages 251–256, 2008.

[24] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[25] Robert M French. The turing test: The first 50 years. *Trends in cognitive sciences*, 4(3):115–122, 2000.

[26] Keinosuke Fukunaga and Larry D Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[27] G David Garson. Computerized simulation in the social sciences: A survey and evaluation. *Simulation & Gaming*, 40(2):267–279, 2009.

[28] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 1–10. Springer, 1999.

[29] Gerd Gigerenzer and Wolfgang Gaissmaier. Heuristic decision making. *Annual review of psychology*, 62:451–482, 2011.

[30] Matthew L Ginsberg and David E Smith. Reasoning about action i: A possible worlds approach. *Artificial intelligence*, 35(2):165–195, 1988.

[31] Sandra S Godfrey and Kenneth R Laughery. The biasing effects of product familiarity on consumers' awareness of hazard. In *Proceedings of the Human Factors Society Annual Meeting*, volume 28, pages 483–486. Sage Publications Sage CA: Los Angeles, CA, 1984.

[32] Karthik Gopalratnam and Diane J Cook. Active lezi: An incremental parsing algorithm for sequential prediction. *International Journal on Artificial Intelligence Tools*, 13(04):917–929, 2004.

[33] Maurits Graafland, Jan M Schraagen, and Marlies P Schijven. Systematic review of serious games for medical education and surgical skills training. *British journal of surgery*, 99(10):1322–1330, 2012.

[34] Ross Graham, Hugh McCabe, and Stephen Sheridan. Pathfinding in computer games. *The ITB Journal*, 4(2):6, 2003.

[35] Andrew Guillory, Hai Nguyen, Tucker Balch, and Charles Lee Isbell Jr. Learning executable agent behaviors from observation. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 795–797. ACM, 2006.

[36] Valerie Guralnik and Karen Zita Haigh. Learning models of human behaviour with sequential patterns. In *Proceedings of the AAAI-02 workshop "Automation as Caregiver"*, pages 24–30, 2002.

[37] Lynne Hall, Sarah Woods, Ruth Aylett, Lynne Newall, and Ana Paiva. Empathic interaction with synthetic characters: the importance of similarity. *Encyclopaedia of Human Computer Interaction*, 2005.

[38] Julianne D Halley and David A Winkler. Classification of emergence and its relation to self-organization. *Complexity*, 13(5):10–15, 2008.

[39] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[40] Philip Hingston. A turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):169–186, 2009.

[41] Jaakko Hintikka. Individuals, possible worlds, and epistemic logic. *Nous*, pages 33–62, 1967.

[42] Nathan Huynh, Rita Snyder, José M Vidal, Omor Sharif, Bo Cai, Bridgette Parsons, and Kevin Bennett. Assessment of the nurse medication administration workflow process. *Journal of healthcare engineering*, 2016.

[43] Nathan Huynh, Rita Snyder, Jose M Vidal, Abbas S Tavakoli, and Bo Cai. Application of computer simulation modeling to medication administration process redesign. *Journal of Healthcare Engineering*, 3(4):649–662, 2012.

[44] Matthew Jack. Tactical position selection. *Game AI Pro: Collected Wisdom of Game AI Professionals*, page 337, 2013.

[45] Jacob Jacoby. Information load and decision quality: Some contested issues. *Journal of Marketing Research*, pages 569–573, 1977.

[46] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[47] Hong Jiang, Jose M Vidal, and Michael N Huhns. Ebdi: An architecture for emotional agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 11. ACM, 2007.

[48] Hong Jiang, José M. Vidal, and Michael N. Huhns. EBDI: An architecture for emotional agents. In *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference*, 2007.

[49] Peter Kaiser, Mike Lewis, Ronald PA Petrick, Tamim Asfour, and Mark Steedman. Extracting common sense knowledge from text for robot planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3749–3756. IEEE, 2014.

[50] Igor V Karpov, Jacob Schrum, and Risto Miikkulainen. Believable bot navigation via playback of human traces. In *Believable Bots*, pages 151–170. Springer, 2012.

[51] Kurt Konolige. What awareness isn't: A sentential view of implicit and explicit belief. In *Proceedings of the 1986 conference on Theoretical aspects of reasoning about knowledge*, pages 241–250. Morgan Kaufmann Publishers Inc., 1986.

[52] Jon A Krosnick. Maximizing questionnaire quality. *Measures of political attitudes*, 2:37–58, 1999.

[53] John Laird and Michael VanLent. Human-level ai's killer application: Interactive computer games. *AI magazine*, 22(2):15, 2001.

[54] Kevin Lam, Babak Esfandiari, and David Tudino. A scene-based imitation framework for robocup clients. *MOO-Modeling Other Agents from Observations*, 2006.

[55] Tijs Leenknegt. Three-dimensional path planning in complex. *Vakgroep Elektronica en Informatiesystemen*, pages 1–80, 2013.

[56] Enrique Leon, Graham Clarke, Victor Callaghan, and Faiyaz Doctor. Affect-aware behaviour modelling and control inside an intelligent environment. *Pervasive and Mobile Computing*, 6(5):559–574, 2010.

[57] Daniel Livingstone. Turing's test and believable ai in games. *Computers in Entertainment (CIE)*, 4(1):6, 2006.

[58] Juan Martinez-Miranda and Arantza Aldea. Emotions in human and artificial intelligence. *Computer and Human Behavior*, 21(2):323–341, 2005.

[59] Marvin Minsky. *Society of Mind*. Simon and Schuster, 1988.

[60] Alexander Nareyek. Game ai is dead. long live game ai! *IEEE intelligent Systems*, 22(1):9–11, 2007.

[61] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta^*: Any-angle path planning on grids. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 1177. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[62] Alex Nash and Sven Koenig. Any-angle path planning. *AI Magazine*, 34(4):85–107, 2013.

[63] Michel Neuhaus and Horst Bunke. Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247, 2007.

[64] Allen Newell and Herbert A Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

[65] Emma Norling, Liz Sonenberg, and Ralph Rönnquist. Enhancing multi-agent based simulation with human-like decision making strategies. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 214–228. Springer, 2000.

[66] Carmine Oliva. Social path following. Master's thesis, ReykjavÃŋk University, 2011.

[67] Justin T Page, Zachary S Liechty, Mark D Huynh, and Joshua A Udall. Bambam: Genome sequence analysis tools for biologists. *BMC Research Notes*, 7(1):829, 2014.

[68] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[69] Bridgette Parsons and José M Vidal. Adaptation and believable human agents. In *SwarmFest Conference Proceedings*, 2015.

[70] Bridgette Parsons, José M Vidal, Nathan Huynh, and Rita Snyder. Automatic generation of agent behavior models from raw observational data. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 121–132. Springer, 2014.

[71] H Van Dyke Parunak, Robert Savit, and Rick L Riolo. Agent-based modeling vs equation-based modeling: A case study and users' guide. *Lecture notes in computer science*, 1534:10–25, 1998.

[72] Rosalind W Picard, Seymour Papert, Walter Bender, Bruce Blumberg, Cynthia Breazeal, David Cavallo, Tod Machover, Mitchel Resnick, Deb Roy, and Carol Strohecker. Affective learningâĂŤa manifesto. *BT technology journal*, 22(4):253–269, 2004.

[73] Steven M Pincus. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, 88(6):2297–2301, 1991.

[74] Helena Rua and Pedro Alvito. Living the past: 3d models, virtual reality and game engines as tools for supporting archaeology and the reconstruction of cultural heritage–the case-study of the roman villa of casal de freiria. *Journal of Archaeological Science*, 38(12):3296–3308, 2011.

[75] Ariel Rubinstein. Similarity and decision-making under risk (is there a utility theory resolution to the allais paradox?). *Journal of economic theory*, 46(1):145–153, 1988.

[76] Javob Schrum, Igor V. Karpov, and Risto Miikkulainen. Ut^2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In *IEEE Conference on Computational Intelligence and Games*, pages 329–336, 2011.

[77] Ben Seymour and Ray Dolan. Emotion, decision making, and the amygdala. *Neuron*, 58(5):662–671, 2008.

[78] G Sharada and OBV Ramanaiah. An artificial intelligence based neuro-fuzzy system with emotional intelligence. *International Journal of Computer Applications*, 1(13):74–79, 2010.

[79] Push Singh. Examining the society of mind. *Computing and Informatics*, 22(6):521–543, 2012.

[80] Aaron Sloman et al. Beyond shallow models of emotion. *Cognitive Processing: International Quarterly of Cognitive Science*, 2(1):177–198, 2001.

[81] Greg Snook. Simplified 3d movement and pathfinding using navigation meshes. *Game Programming Gems*, 1(1):288–304, 2000.

[82] Rita Snyder, Nathan Huynh, Bo Cai, José Vidal, and Kevin Bennett. Effective healthcare process redesign through an interdisciplinary team approach. In *Studies in Healthcare Technology and Informatics*, volume 192. MEDINFO 2013, 2013.

[83] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: A structural description of the human brain. *PLoS Comput Biol*, 1(4):e42, 2005.

[84] Harold Stanislaw. Tests of computer simulation validity: What do they measure? *Simulation & Games*, 1986.

[85] Gurkan Tuna, Tarik Veli Mumcu, Kayhan Gulez, Vehbi Cagri Gungor, and Hayrettin Erturk. Unmanned aerial vehicle-aided wireless sensor network deployment system for post-disaster monitoring. In *International Conference on Intelligent Computing*, pages 298–305. Springer, 2012.

[86] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[87] Iskander Umarov and Maxim Mozgovoy. Believable and effective ai agents in virtual worlds: Current state and future perspectives. *International Journal of Gaming and Computer-Mediated Simulations (IJGCMS)*, 4(2):37–59, 2012.

[88] Niels Van Hoorn, Julian Togelius, Daan Wierstra, and Jürgen Schmidhuber. Robust player imitation using multiobjective evolution. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 652–659. IEEE, 2009.

[89] Juan D Velásquez. Modeling emotions and other motivations in synthetic agents. In *AAAI/IAAI*, pages 10–15. Citeseer, 1997.

[90] Monica L Visinsky, Joseph R Cavallaro, and Ian D Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering & System Safety*, 46(2):139–158, 1994.

[91] Joachim Wahle and Michael Schreckenberg. A multi-agent system for on-line simulations based on real-world traffic data. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, pages 9–pp. IEEE, 2001.

[92] Joseph Weizenbaum. Eliza – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.